Daniel Mendez
Paris Avgeriou
Marcos Kalinowski
Nauman Bin Ali *Editors*

Handbook on Teaching Empirical Software Engineering

Springer

Handbook on Teaching Empirical Software Engineering

Daniel Mendez • Paris Avgeriou •
Marcos Kalinowski • Nauman Bin Ali
Editors

Handbook on Teaching Empirical Software Engineering

Springer

Editors
Daniel Mendez
Blekinge Institute of Technology and fortiss
Karlskrona, Sweden

Paris Avgeriou
University of Groningen
Groningen, The Netherlands

Marcos Kalinowski
Pontifical Catholic University of Rio de Janeiro
Rio de Janeiro, Brazil

Nauman Bin Ali
Blekinge Institute of Technology
Karlskrona, Sweden

ISBN 978-3-031-71768-0 ISBN 978-3-031-71769-7 (eBook)
https://doi.org/10.1007/978-3-031-71769-7

© The Editor(s) (if applicable) and The Author(s), under exclusive license to Springer Nature Switzerland AG 2024

This work is subject to copyright. All rights are solely and exclusively licensed by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.
The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.
The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

If disposing of this product, please recycle the paper.

Foreword by Vic Basili

I like to think that empirical software engineering has put the science into computer science. It has enabled the examination and evaluation of theories and practices in the development of software. I am reminded of a review of a paper that I published in 1981 with Robert Reiter, entitled "A Controlled Experiment Quantitatively Comparing Software Development Approaches" in the *IEEE Transactions on Software Engineering*, which was awarded the IEEE Computer Society Outstanding Paper Award. The reviewer said he didn't understand why we had to run the experiment as he would have guessed the results of the experiment. The review surprised me as it is one thing to have an opinion and another to have evidence of what works and what does not work and maybe why it works. Empirical software engineering includes everything from estimating the cost of a software development to evaluating the quality of a software product to evaluating the various techniques used to develop software in the first place.

The growth of the field of empirical software engineering has been phenomenal. The first ISERN Conference, with the aim of creating a forum for empirical software engineering, began in 1992 created by Rombach, Oivo, and myself, and the *Journal of Empirical Software Engineering*, created by Warren Harrison and myself, was first published by Kluwer Academic Publishers in 1996 with a page count of 305 pages. It was taken over by Springer in 2005, with Lionel Briand and myself as co-editors. That journal is now well over 3000 pages a year.

One side note about the empirical software engineering community, based upon my attendance at the 30 th anniversary of ISERN in October of 2023, is that the community is inclusive and collaborative. I was impressed by the level of support, camaraderie, and collaboration among the various members of the community.

And so, it is clearly time for a textbook to present in one place, the variety of options for supporting a curriculum in empirical software engineering. Although there are a great number of papers that provide specific experimental results relating to empirical software engineering, there has not been a textbook that can be used to teach how to do it and provide in one place a collection of pedagogical approaches and related resources.

This book covers many relevant topics for teaching empirical software engineering from instrumentation and measurement, e.g., action research, survey research, qualitative data analysis, experimental design, case study research, systematic literature reviews, and mining software repositories.

It provides a collection of techniques and approaches that concentrate and organize support for empirical software engineering education.

University of Maryland Victor R. Basili
USA

Foreword by Claes Wohlin

Software engineering is a human-intensive engineering discipline requiring other research methods than most other engineering disciplines, including computer science. Consequently, empirical research has become an essential part of contemporary software engineering.

Empirical studies are helpful for both practitioners and researchers. Practitioners may evaluate how users perceive different solutions, for example, Web page designs or alternative feature implementations. Moreover, practitioners may want to assess whether a new method, technique, or tool improves what is currently used. Researchers may use empirical methods to explore, describe, explain, or evaluate a software engineering activity or artifact, for example, studying an activity at a company using case study or action research, or evaluating a novel tool for software testing. The different objectives for launching an empirical study imply that software engineering practitioners and researchers need many research methods in their research toolbox.

Taking an evolutionary perspective on software engineering, empirical studies were rare in the early days of software engineering research. However, as time passed, researchers began promoting software engineering as an experimental science. In the 1990s, it was enlarged into an empirical science, and it was established as such through the creation of conferences and a journal devoted to empirical software engineering. Furthermore, other conferences and journals also started encouraging and welcoming empirical research articles. Around and after the millennium, books dedicated to empirical research in software engineering were published. They include books focusing on specific research methods, primarily experimentation, case study research, and action research, as well as edited books covering different aspects of empirical software engineering research. However, the primary focus has, so far, been on conducting research.

Based on the learning from conducting empirical research in software engineering, it is essential to further integrate empirical research into the software engineering curricula at the different levels of software engineering education. The courses may be general courses concerning empirical research methods or focusing on specific aspects of empirical research, such as research ethics. Moreover,

empirical studies can be a natural part of the thesis work for bachelor, master, and PhD level studies. As we continue to integrate empirical research methods and studies into education, it becomes essential to have good sources for teaching empirical research methods and to learn how to conduct empirical studies in software engineering.

The current book, *Teaching Empirical Research Methods in Software Engineering*, edited by Professors Daniel Mendez, Paris Avgeriou, Marcos Kalinowski, and Dr. Nauman Bin Ali, takes a timely focus on teaching empirical software engineering. The editors have done an excellent job attracting some of the area's world-leading researchers and teachers. The book provides an excellent collection of chapters covering different empirical research methods. Furthermore, the chapters take a pedagogical stance concerning several essential topics to teach. The focus on teaching is well illustrated by the titles of many of the chapters.

The book is highly recommended for teachers who want to integrate empirical research methods into their courses at different educational levels or run a separate course on empirical research methods. Furthermore, the book is also recommended for students who wish to become more proficient in conducting empirical studies.

Blekinge Institute of Technology Claes Wohlin
Karlskrona, Sweden

Foreword by Guilherme Travassos

Teaching is one of the most exciting activities for me. The opportunity to share knowledge with people makes me feel good and energized. However, teaching can be cumbersome when we work with emergent and crucial topics such as experimental software engineering, which is paramount in our field. The complexity of this topic underscores the need for more in-depth understanding and discussion.

It must be recognized that systematically, in the last 40 years, efforts have been made to provide information on planning, executing, analyzing, and reporting different empirical studies, which supported the organization of courses on empirical research. The International Software Engineering Research Network (ISERN) is the most representative example of this.[1] However, it must be observed that little information was made available on how to teach empirical and experimental research despite the concerns raised at the Dagstuhl seminars (9238, 06262,[2] 10122[3]) in 1992, 2006, and 2010.

Likely, the lack of teaching materials on empirical research challenged us when introducing the courses in experimental software engineering in our universities, most of them showing up after the Dagstuhl seminar in 1992. For instance, at the Federal University of Rio de Janeiro, the first experimental software engineering course was held in 2001,[4] discussing experimental research methods, methodology

[1] Felderer, M. and Travassos, G.H. (2020). The Evolution of Empirical Methods in Software Engineering. Contemporary Empirical Methods in Software Engineering. Springer. https://doi.org/10.1007/978-3-030-32489-6_1

[2] https://www.dagstuhl.de/seminars/seminar-calendar/seminar-details/06262

[3] https://www.dagstuhl.de/en/seminars/seminar-calendar/seminar-details/10122

[4] Rainer, A.; Ciolkowski, M.; Pfhal, D. et al. (2006), Teaching Empirical Methods to Undergraduate Students Working Group Results. LNCS 4336: Empirical Software Engineering Issues – Critical Assessment and Future Directions. Eds. Basili, Rombach, Schneider, Kitchenham, Pfahl, Selby. Springer. ISSN 0302-9743

of experimental studies, data representation, analysis and interpretation, and statistical methods regarding quantitative primary studies.

Since its first edition, much has evolved in the annual editions. Other courses needed to be organized over the years to deal with the number of empirical research opportunities available to support the scientific method in software engineering through qualitative studies, case studies, action research, simulated-based studies, systematic literature studies, aggregation, and evidence synthesis. I must recognize that the tacit knowledge of previous teaching and the participation in the organization of some of these contemporary empirical strategies helped me a lot in organizing the courses.

However, only those who took part in these courses could testify to the mistakes and improve their courses, which are unlikely to follow a similar syllabus, possibly creating a teaching gap. Besides, the lack of training material and the missing of empirical methods discussions in the main venues can be an explanation for many misunderstandings about empirical strategies, such as case studies, systematic literature reviews, measurement, and statistical methods that we can observe in some technical works due to the difficulty or lack of time of the prospective researchers to learn empirical research correctly.

Consider this book a golden opportunity! I am thrilled to have the honor of writing a foreword for this exceptional edited book on *Teaching Empirical Research Methods in Software Engineering*. It stands out among the other textbooks on classical and contemporary empirical methods, offering a unique and valuable resource for introducing and teaching empirical research in software engineering.

The editors Daniel Mendez, Paris Avgeriou, Marcos Kalinowski, and Nauman Bin Ali, together with a very competent team of authors, offer explicit knowledge of relevant and exciting experiences and topics to support the introduction and teaching of empirical research in software engineering.

It is worth reading!

The collective effort of the contributors fills me with joy. It demonstrates the scientific competence, plurality, and multidisciplinarity of knowledge in our field and reflects the strong sense of community and collaboration so valuable and necessary in empirical and experimental software engineering.

As I used to say to my experimental software engineering team, anxiety is the enemy of science. This is yet more evidence that good advancements take time.

This book is evidence of knowledge maturity and adequate initiative toward the evolution of teaching experimentation in software engineering.[5]

Enjoy the reading!

Federal University of Rio de Janeiro Guilherme Horta Travassos
Brazil

[5] Travassos, G. H. (2006). From Silver Bullets to Philosophers´ Stones: Who Wants to Be Just an Empiricist? LNCS 4336: Empirical Software Engineering Issues – Critical Assessment and Future Directions. Eds. Basili, Rombach, Schneider, Kitchenham, Pfahl, Selby. Springer. ISSN 0302-9743

Preface

Empirical software engineering has received much attention in recent years and helped shift software engineering research from a more mathematically oriented and design science-driven discipline to a more insight-oriented and theory-driven one. We may well argue that through that shift, our discipline has evolved and grown significantly in terms of scientific maturity, and so has the need for enforcing the highest standards in empiricism. While extensive guidelines are nowadays available for designing, conducting, reporting, and reviewing empirical studies, similar attention has not yet been paid to *teaching* empirical software engineering. Closing this gap is the scope of this edited book.

In this book, we draw from the profound experience and expertise of well-established scholars in the empirical software engineering community to provide guidance and support in teaching various research methods and fundamental concepts. The main audience of this book includes, but is not limited to:

1. Educators teaching empirical software engineering to undergraduate, postgraduate, or doctoral students in higher education settings
2. Professional trainers teaching the basic concepts of empirical software engineering to software professionals or other lifelong learners
3. Students and trainees following these courses

Our book is organized into four major parts: In an *editorial introduction*, we, the editors, set the foundation with two chapters—one laying out the larger context of the discipline for a positioning of the remainder of this book, and one guiding the creation of a syllabus for courses in empirical software engineering. The second part of this book lays the *fundamentals* for teaching empirical software engineering, addressing more cross-cutting aspects from theorizing and teaching research designs, to measurement and quantitative data analysis. In the third part, our contributing authors share general *experiences and personal reflections* from teaching empirical software engineering. Finally, the fourth part contains a number of carefully *selected research methods,* presented through an educational lens.

Next to the chapter contributions themselves that provide a more theoretical perspective and practical advice, readers will find additional material in the form

of, for example, slide sets and tools, in our online material section. That online material section is expected to be updated regularly and well beyond the publication date of this book. It is centrally available through www.emse.education as well as a permanently archived, synchronized, and citable Zenodo release under https://zenodo.org/doi/10.5281/zenodo.11544897.

It goes without saying that this book would not have been possible without the excellent contributions of the community and the immense collegial support in the form of continuous exchanges, peer reviews, and fruitful discussions. We are therefore profoundly thankful to all contributing authors and hope that this book will further strengthen the educational impact of the community.

Karlskrona, Sweden — Daniel Mendez
Groningen, The Netherlands — Paris Avgeriou
Rio de Janeiro, Brazil — Marcos Kalinowski
Karlskrona, Sweden — Nauman Bin Ali

Contents

Part I Editorial Introduction

Teaching Empirical Software Engineering: An Editorial Introduction 3
Daniel Mendez, Paris Avgeriou, Marcos Kalinowski, and Nauman Bin Ali

Designing a Syllabus for a Course on Empirical Software Engineering ... 13
Paris Avgeriou, Nauman Bin Ali, Marcos Kalinowski, and Daniel Mendez

Part II Fundamentals for Teaching Empirical Software Engineering

Teaching Theorizing in Software Engineering Research 31
Klaas-Jan Stol

Teaching Research Design in Software Engineering 71
Jefferson Seide Molléri and Kai Petersen

Teaching Software Metrology: The Science of Measurement for Software Engineering 101
Paul Ralph, Miikka Kuutila, Hera Arif, and Bimpe Ayoola

Qualitative Data Analysis in Software Engineering: Techniques and Teaching Insights 155
Christoph Treude

Part III Experience Reports and Reflections

Teaching Empirical Methods at Eindhoven University of Technology 179
Alexander Serebrenik and Nathan Cassee

Ten Years of Teaching Empirical Software Engineering in the Context of Energy-Efficient Software 209
Ivano Malavolta, Vincenzo Stoico, and Patricia Lago

Experiences in Using the V-Model as a Framework for Applied Doctoral Research 255
Rodrigo Falcão, Andreas Jedlitschka, Frank Elberzhager, and Dieter Rombach

Sustainability Competencies in Research Strategy for Software Engineering—Personal Experience and Best Practices 277
Birgit Penzenstadler

Part IV Teaching Empirical Software Engineering Research Methods

A Course on Experimentation in Software Engineering: Focusing on Doing 301
Sira Vegas and Natalia Juristo

Teaching Mining Software Repositories 325
Zadia Codabux, Fatemeh Fard, Roberto Verdecchia, Fabio Palomba, Dario Di Nucci, and Gilberto Recupito

Teaching Simulation as a Research Method in Empirical Software Engineering 363
Breno Bernard Nicolau de França, Dietmar Pfahl, Valdemar Vicente Graciano Neto, and Nauman Bin Ali

Teaching Action Research 387
Miroslaw Staron

Action Research with Industrial Software Engineering: An Educational Perspective 413
Yvonne Dittrich, Johan Bolmsten, and Catherine Seidelin

Teaching Case Study Research 463
Stefan Wagner

Teaching Design Science as a Method for Effective Research Development 479
Oscar Pastor, Mmatshuene Anna Segooa, and Jose Ignacio Panach

Teaching Survey Research in Software Engineering 501
Marcos Kalinowski, Allysson Allex Araújo, and Daniel Mendez

Teaching Literature Reviewing for Software Engineering Research 529
Sebastian Baltes and Paul Ralph

Teaching Systematic Literature Reviews: Strategies and Best Practices 557
Marcela Genero and Mario Piattini

Teaching and Learning Ethnography for Software Engineering Contexts 593
Yvonne Dittrich, Helen Sharp, and Cleidson de Souza

Contributors

Nauman Bin Ali Blekinge Institute of Technology, Karlskrona, Sweden

Allysson Allex Araújo Federal University of Cariri, Juazeiro do Norte, Brazil

Hera Arif Dalhousie University, Halifax, NS, Canada

Paris Avgeriou University of Groningen, Groningen, The Netherlands

Bimpe Ayoola Dalhousie University, Halifax, NS, Canada

Sebastian Baltes University of Bayreuth, Bayreuth, Germany

Johan Bolmsten World Maritime University, Malmö, Sweden

Nathan Cassee Eindhoven University of Technology, Eindhoven, The Netherlands

Zadia Codabux University of Saskatchewan, Saskatoon, SK, Canada

Breno Bernard Nicolau de França Breno Bernard Nicolau de França, UNICAMP, Campinas, Brazil

Cleidson de Souza Universidade Federal do Pará: Belém, Belém, Pará, Brazil

Dario Di Nucci University of Salerno, Salerno, Italy

Yvonne Dittrich IT University of Copenhagen, Copenhagen, Denmark

Frank Elberzhager Fraunhofer IESE, Kaiserslautern, Germany

Rodrigo Falcão Fraunhofer IESE, Kaiserslautern, Germany

Fatemeh Fard University of British Columbia, Kelowna, BC, Canada

Marcela Genero Department of Technologies and Information Systems, University of Castilla-La Mancha, Ciudad Real, Spain

Andreas Jedlitschka Fraunhofer IESE, Kaiserslautern, Germany

Natalia Juristo Universidad Politécnica de Madrid, Madrid, Spain

Marcos Kalinowski Pontifical Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil

Miikka Kuutila Dalhousie University, Halifax, NS, Canada

Patricia Lago Vrije Universiteit Amsterdam, Amsterdam, The Netherlands

Ivano Malavolta Vrije Universiteit Amsterdam, Amsterdam, The Netherlands

Daniel Mendez Blekinge Institute of Technology and fortiss, Karlskrona, Sweden

Jefferson Seide Molléri Kristiania University College, Oslo, Norway

Fabio Palomba University of Salerno, Salerno, Italy

Jose Ignacio Panach Escola Tècnica Superior d'Enginyeria, Universitat de València, Valencia, Spain

Oscar Pastor Research Center on Software Production Methods (PROS), Universitat Politècnica de València, Valencia, Spain

Birgit Penzenstadler Chalmers University of Technology and University of Gothenburg, Gothenburg, Sweden
Lappeenranta Lahti University, Lappeenranta, Finland

Kai Petersen Flensburg University of Applied Sciences, Flensburg, Germany
Blekinge Institute of Technology, Karlskrona, Sweden

Dietmar Pfahl University of Tartu, Tartu, Estonia

Mario Piattini Department of Technologies and Information Systems, University of Castilla-La Mancha, Ciudad Real, Spain

Paul Ralph Dalhousie University, Halifax, NS, Canada

Gilberto Recupito University of Salerno, Salerno, Italy

Dieter Rombach RPTU Kaiserslautern, Kaiserslautern, Germany

Mmatshuene Anna Segooa Informatics Department, Tshwane University of Technology, Faculty of ICT, Pretoria, South Africa

Catherine Seidelin Novo Nordisk, Bagsværd, Denmark

Alexander Serebrenik Eindhoven University of Technology, Eindhoven, The Netherlands

Helen Sharp The Open University, Milton Keynes, UK

Miroslaw Staron University of Gothenburg, Gothenburg, Sweden

Vincenzo Stoico Vrije Universiteit Amsterdam, Amsterdam, The Netherlands

Klaas-Jan Stol Lero and University College Cork, Cork, Ireland

Christoph Treude Singapore Management University, Singapore, Singapore

Sira Vegas Universidad Politécnica de Madrid, Madrid, Spain

Roberto Verdecchia University of Florence, Firenze, Italy

Valdemar Vicente Graciano Neto Universidade Federal de Goiás, Goiânia, Brazil

Stefan Wagner TUM School of Communication, Information and Technology, Technical University of Munich, Heilbronn, Germany

Part I
Editorial Introduction

Teaching Empirical Software Engineering: An Editorial Introduction

Daniel Mendez, Paris Avgeriou, Marcos Kalinowski, and Nauman Bin Ali

Abstract Empirical software engineering has received much attention in recent years and became a de-facto standard for scientific practice in software engineering. However, while extensive guidelines are nowadays available for designing, conducting, reporting, and reviewing empirical studies, similar attention has not yet been paid to teaching empirical software engineering. Closing this gap is the scope of this edited book.

In the following editorial introduction, we, the editors, set the foundation by laying out the larger context of the discipline for a positioning of the remainder of this book.

1 Introduction

Empirical software engineering has received much attention in recent years and marked a certain shift from a more mathematically oriented and design science-driven engineering discipline to a more insight-oriented and theory-driven one [12]. Before that, a large extent of the available body of knowledge in software engineering was rooted in rationalist arguments where "common knowledge" was established on the sole ground of reasoning by argument (and, in some cases,

D. Mendez (✉)
Blekinge Institute of Technology and fortiss, Karlskrona, Sweden
e-mail: daniel.mendez@bth.se

P. Avgeriou
University of Groningen, Groningen, The Netherlands
e-mail: p.avgeriou@rug.nl

M. Kalinowski
Pontifical Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil
e-mail: kalinowski@inf.puc-rio.br

N. B. Ali
Blekinge Institute of Technology, Karlskrona, Sweden
e-mail: nauman.ali@bth.se

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2024

D. Mendez et al. (eds.), *Handbook on Teaching Empirical Software Engineering*,
https://doi.org/10.1007/978-3-031-71769-7_1

reasoning by authority), logical inference, and mathematical proofs. While this may still be the case for certain areas within software engineering, our discipline as a whole has evolved and grown significantly in terms of scientific maturity; so has the need for excellence in empiricism, as we now reason, in simple terms, by evidence gained through sensory experiences[1] (e.g. observations) while following certain rules, principles, and norms for scientific practice [16].

The ambition is that decisions in software engineering practice become rooted in empirical evidence [1, 11]. Those decisions may well range from the choice of software engineering methods, modelling and description techniques, and tools over to testing the sensitivity of the software-intensive products and services themselves to the characteristics and needs of their practical contexts (i.e. fit for purpose). The importance of empirical software engineering research[2] is, therefore, well recognised by both researchers and practitioners, and competencies in empirical software engineering are in high demand.

It is therefore not surprising that more empirically oriented forums like the Empirical Software Engineering journal (EMSE), the International Symposium on Empirical Software Engineering and Measurement (ESEM) and the International Software Engineering Research Network (ISERN), or the International Conference on Evaluation and Assessment in Software Engineering (EASE) have become part of the prestigious venues in the software engineering research community [18]. Moreover, even the more traditional, leading venues like IEEE Transactions on Software Engineering (TSE) or the International Conference on Software Engineering (ICSE) have come to expect an empirical evaluation of proposed ideas as a de facto prerequisite to merit the inclusion of manuscripts in their programmes.

To become a reflective software engineer [7], practitioners need to be equipped with the abilities and skills for critical thinking and reasoning, systematic problem-solving, and objective decision-making not only based on reasoning by (rationalist) arguments but also based on empirical insights and data. In fact, leveraging empirical software engineering competencies is a prerequisite for continuous improvement [3, 6] as they enable organisations to test new ideas, processes, and technologies, fostering a learning environment where they can adapt and evolve based on empirical data [3, 6]. High software development maturity has consequently been related to establishing empirical and data-driven continuous improvement capacities to ensure that innovations align with the business goals and contribute to enhanced quality and process performance [4, 10].

The demand for competencies in empirical software engineering gave rise to various courses on empirical software engineering in both higher education and

[1] We borrow the term "sensory experiences" from the established terminology in the philosophy of science [16] and subsequently use it in the widest possible sense. As software artefacts and phenomena can materialise in various ways, the term may well include direct and indirect observations (e.g. via measurements).

[2] In the remainder of this chapter, we refer with *empirical software engineering* to the research activities aiming at producing evidence (i.e. conducting and reporting empirical studies) rather than to the act of consuming evidence.

professional training programmes. The goals of these courses range from preparing students for their graduate programmes and thesis projects to providing them with essential skills for a career in software engineering research and practice.

While extensive guidelines are available for designing, conducting, reporting, and reviewing empirical studies, similar attention has not yet been paid to teaching empirical software engineering. We, therefore, recognise that there is a need to gather experiences, best practices, and lessons learnt from the community to jointly develop and inform each other on possibilities for effectively teaching the principles and methods of empirical software engineering. Nowadays, courses on empirical software engineering are mostly designed independently (thus, reinventing the wheel every time and sometimes proposing competing or even conflicting concepts and ideas) without relying on a shared basis for knowledge, materials, and what could be considered good practices.

Closing this gap sets the aim and scope of this edited book. Before introducing the scope of the book, we attempt to motivate the challenges in empirical software engineering through the lenses of educators. This, in turn, requires a basic understanding of what empirical software engineering is, which we address next.

2 Empirical Software Engineering in a Nutshell

As empirical software engineering may come with different interpretations, also due to the lack of a commonly used definition, we introduce the very basic concepts and terms next.

Software engineering research and practice are continuously evolving to cope with the emerging challenges imposed by the ubiquitous nature and rapid advancements in engineering methods and technologies used. This poses various challenges for the selections and configurations of engineering teams, methods, and tools that can no longer be answered by rationalist arguments only.

In response to the emerging challenges in software engineering, we have already observed a prominent turn towards empirical approaches. On the one hand, the recognition of the importance of empirical research to software engineering allows us scholars to critically reflect upon central and, to some extent, more philosophical questions like "What qualifies as scientifically sound practice?" or "What are relevant research questions and appropriate research methods to address a certain research problem?". On the other hand, it has also yielded various more practical contributions that impact software engineering practice.

The community of researchers has already made strong contributions that allowed us to shift from a purely design science-oriented discipline driven by the application of scientific concepts and methods to practical ends—rendering the discipline as a more traditional *engineering* discipline—to a more insight-oriented and theory-driven (*scientific*) discipline [17].

This shift towards a more insight-oriented discipline based on empirical evidence has taken place in recent years with a growing and vibrant community of researchers coining it as *empirical software engineering (research)* (EMSE) [2, 11, 15].

In simple terms and for the sake of this brief introduction, the ultimate goal of empirical software engineering is to advance our body of knowledge by continuously building and evaluating scientific theories.

Scientific Knowledge and Theories
With *scientific knowledge*, we refer to a portrait of our (human) understanding of reality which is captured via scientific theories.

With *scientific theory*, we refer to the belief that there are patterns in phenomena while having survived (1) tests against sensory experiences and (2) criticism by critical peers. With the first criterion, we refer to what is commonly known as the demarcation problem introduced by Karl Popper [13]. Here, the main argument is that a theory qualifies only as a scientific theory (rather than a non-scientific or a pseudoscientific one), if the underlying hypotheses are testable and, most importantly, falsifiable. With the second criterion, we refer to quality assurance of scientific practices and safeguards implemented by the research community, such as peer review. This may of course render the acceptance of scientific results also as a social process.

Empirical software engineering has relevance to both academic research communities and software engineering practice alike, as it allows us, eventually, to:

1. Reason about our discipline and the involved phenomena (e.g. social phenomena)
2. Recognise and understand the limitations and effects of artefacts in their practical contexts (e.g. by evaluating technologies, techniques, processes, or models)

To picture a process, if there is any at all that could characterise the general way of working in empirical software engineering, we believe that it is best described by what we introduced as a pragmatic cycle, as shown in Fig. 1 and taken from [12]. That cycle characterises in its essence the different perspectives we may take on the notion of theories in software engineering and consequently the approaches we may take towards building and evaluating theories:

- *Induction* describes the inference of a general rule from a particular case and is often referred to as the principle of building/extending theories, for example, by conducting observational studies to explore practical, real-life contexts and relevant parameters.
- *Deduction* describes the application of a general rule to a particular case and is often referred to as theory evaluation, where we test whether the hypothesised outcome of a theory can also be observed in reality, for example, by conducting quasi-controlled experiments to corroborate concepts by testing the expected improvements.

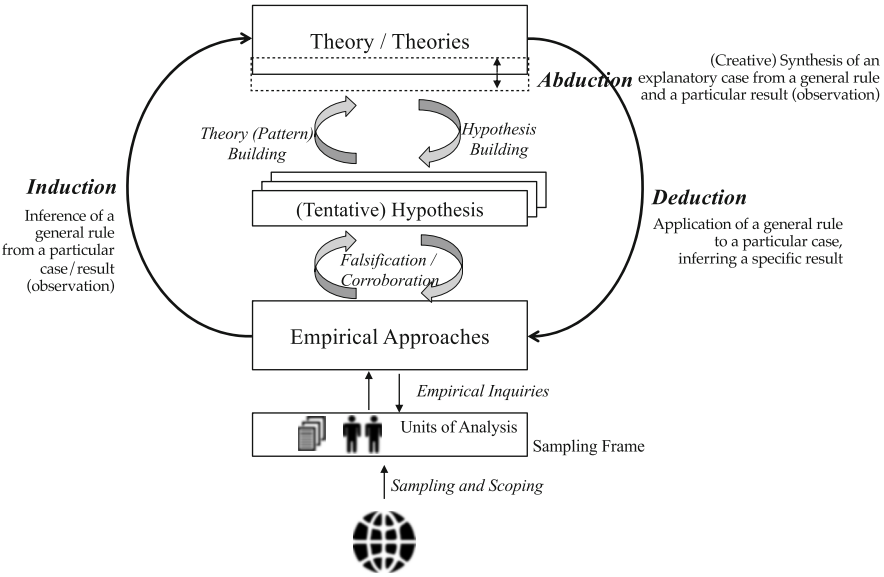


Fig. 1 Cycle for empirical research, taken from [12]

- *Abduction* describes the, to some extent, creative synthesis of a case from a general rule and a particular result and used, for example, to infer probable cases out of existing theories.

In empirical software engineering, assuming an ideal scenario, we would follow that cycle multiple times to contextualise and increase the robustness of our theories and, eventually, scale them up to practice by triangulating through different settings and methods (both quantitative and qualitative ones).

While we refer to [9, 12] for a more profound discussion of empirical software engineering and its historical evolution, we may conclude here, for the sake of our introductory chapter, that every empirical approach and every epistemological stance,[3] of which there are many, has its well-deserved place in that larger picture that has been elaborated under the assumption of the following postulates:

1. There is nothing absolute about truth in the notion of scientific theories, but good (commonly accepted) scientific principles and norms to build and evaluate theories and, thus, increase our confidence (level of evidence).
2. There is no such thing as a universally valid way of working; in fact, there is no rigid, let alone standardised, process but many different and valid (and complementary) ways of undertaking research.

[3] With epistemology, we refer to the theory of knowledge.

3. There is no single, empirically inquired point of view that will ever provide us with an entire picture when interpreting relevant real-world phenomena.

As a consequence of the above, elaborating strong, robust theories in the community—that is, theories with a high level of confidence—also requires multiple iterations through the cycle (e.g. with replication studies) using complementary research methods while constantly setting the result of each inquiry in relation to existing evidence.

We therefore recognise well that there is no purity in empirical software engineering and no single empirical approach that would be best for all purposes. At the same time, we have still reached a common understanding in the community of researchers of the basic concepts and terms, and we can currently observe already excellent contributions to research and practice. This is also corroborated by edited collections in our community that contribute to laying out more fundamental positionings, experience-based recommendations and guidelines, and general approaches to conduct empirical software engineering research, such as [8]. These are further complemented by attempts to standardising quality criteria of reported studies [14].

The question that remains now open is how to educate and train students and software practitioners in those concepts. More specifically, there are certain challenges in doing so, as elaborated in the following section.

3 Challenges when Teaching Empirical Software Engineering

When teaching empirical software engineering, we face no fewer challenges than when engaging in empiricism in research and practice. We believe that among the most pressing challenges that educators face are the following three:

Lack of Purity Software engineering is to some extent also a social practice [5] and as such it incorporates also the same plurality as the social sciences. In that sense, there is no purity in empirical software engineering research, same as there is no purity in software engineering practice. Software engineering research and practice are characterised by many different families of systems, domains, subject areas, and disciplinary backgrounds, many coming from natural sciences. This has effects on preferred principles and understanding, often coming with some forms of prejudice (e.g. artificially separating qualitative from quantitative camps). As no empirical approach fits best for all purposes, students should be taught how to choose or combine suitable methods based on research goals and context and adapt them based on reflective evaluations of outcomes. Embracing this lack of a standardised way of working is, therefore, a fundamental virtue to empirical research in software engineering, but this makes teaching it inherently challenging. With chapters taking a broader perspective on the variety of research methods and their combination, such

as the chapter on research designs in software engineering, we attempt to (partially) address this challenge.

Teaching EMSE Requires More Than Teaching the Details of Single Research Methods While we may well agree on basic concepts and terminology, at least in the long run, we will still be confronted with the limitations given in classroom settings where students will have difficulties experiencing and understanding the core of empirical software engineering (research). This is also because empirical software engineering goes well beyond the scope of a single more traditional research discipline in, for example, natural sciences. It requires a more profound understanding of its philosophical roots and its practical implications. For example, how to appropriately frame a research problem, how to identify relevant research questions, what relevant means both from a theoretical and a practical perspective, and how to choose and apply research methods and adequate data analysis techniques make clear that teaching EMSE goes beyond teaching single research methods but requires a profound understanding of a larger picture where philosophical arguments and trade-offs in the choice of research methods are the norm. This is especially true when scaling method application and analysis. We try to address this challenge, at least to some extent, with chapters focusing more on the fundamentals of empirical software engineering, such as on instrumentation and measurements, or on theorising in software engineering research.

Difficulty in Teaching Flexible Design Studies Teaching flexible design research is additionally hard in traditional courses because (a) the design of the study evolves based on the findings, (b) it is inconceivable even to foresee all design decisions that will need to be taken, and (c) it often involves qualitative data collection and analysis. These characteristics make it difficult to experience such research in a course. On the contrary, for a fixed design study involving an experiment or a questionnaire-based survey with quantitative data, the students can experience the challenges of designing a complete study and, through pilots, even experience operationalising and reporting the study. With several chapters on teaching flexible design studies like action research, case studies, and ethnographies, this book aims to support educators in teaching these particularly challenging methods.

The difficulty we faced when planning this book project was, therefore, to balance the need of recognising the above outlined diversity in empirical software engineering while still preserving some common ground on which we all can agree.

4 Scope of the Book

This book draws from the profound experience and expertise of well-established scholars in the empirical software engineering community to provide guidance and support in teaching the various approaches and philosophical stances.

A particular focus is on combining (1) research methods and their epistemological settings and terminology and the (2) didactics and pedagogy for the

subject. The book covers the most essential and contemporary research methods and philosophical and cross-cutting concerns in software engineering research, considering both academic and industrial settings. It also provides insights into the effective teaching of concepts and strategies.

Rather than providing a set of prescriptive blueprints for blindly setting up courses, our ambition was to provide the readers with both a more generic (customisable) basis for course design and practical advice and support; the latter comes in the form of experience reports, best practices, annotated bibliographies/readers, slide decks, datasets, available tools, prerequisite knowledge, examples/exercises (formative), and assignments—found in the subsequent chapters and the online material sections. This can equip educators with the necessary information and tools for effectively preparing students for today's prevalent needs in software engineering research and practice.

Online Repository for this Edited Book
Many of the chapters refer to online material including, if not limited to, bibliographies/readers, slide decks, datasets, tools, examples/exercises, or assignments. This repository is centrally available through www.emse.education and may be updated regularly. A permanently archived, synchronised, and citable Zenodo release can be found under https://zenodo.org/doi/10.5281/zenodo.11544897.

This edited book at hand is therefore meant to (1) provide a common foundation for educators while (2) recognising (and cherishing) the broad spectrum of possible flavours. We hope to have achieved this by approaching this edited book as a joint community project right from its conception. The idea for this book was born as part of a community workshop conducted at the International Software Engineering Research Network (ISERN) and discussed further with members of the research community before yielding an open call for contributions. The editorial and peer review process was further facilitated by that community with frequent exchanges, feedback, and collegial support.

We take special pride in the outcome that constitutes a collection of topics covering the broad spectrum of empirical software engineering we believe is important to be represented in educational settings and written by a variety of internationally well-established scholars.

In this book, we distinguish essentially two major categories of chapters:

1. Chapters reflecting upon more cross-cutting concepts, such as measurement or mining data repositories, as well as chapters offering teaching experience reports
2. Chapters emphasising the sharing of experience-based advice for effectively teaching selected empirical methods

Finally, in both categories, we distinguish between more experience-based reflections that may well come with outlines of entire courses and a reflection of lessons learnt and more foundational chapters that are meant to support readers in setting up their own courses by providing the necessary knowledge and advice. We trust that the reader will find both categories of chapters useful and that they will know how to take both as orientation when designing their own courses.

In the second editorial chapter described next, we introduce a community-inspired syllabus and general reflection upon general contents when teaching empirical research methods. There, we also set the chapter contributions into context to provide an outline of the edited book.

Online Material

In our online material repository https://zenodo.org/doi/10.5281/zenodo.11544897, the reader can find different slide decks that introduce the topic area of empirical software engineering in different educational settings, each having with special emphasis on building and evaluating theories.

Acknowledgments We would like to express our profound gratitude to all authors for the various thought-provoking and fruitful exchanges, their feedback and peer reviews, and their valuable contributions, all making this edited book on Teaching Empirical Software Engineering possible. We would also like to thank Yvonne Dittrich for inspiring exchanges and invaluable feedback on earlier versions of this particular chapter.

References

1. Ali, N.B.: Is effectiveness sufficient to choose an intervention? Considering resource use in empirical software engineering. In: Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM 2016, Ciudad Real, Spain, September 8–9, 2016, pp. 54:1–54:6. ACM (2016). https://doi.org/10.1145/2961111.2962631
2. Basili, V.: A personal perspective on the evolution of empirical software engineering. In: Muench, J., Schmid, K. (eds.) Perspectives on the Future of Software Engineering, pp. 255–273. Springer, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37395-4_17
3. Basili, V.R., Caldiera, G., Rombach, H.D.: The experience factory. In: Marciniak, J.J. (ed.) Encyclopedia of Software Engineering, pp. 469–476. Wiley, New York, NY, USA (1994)
4. CMMI Institute: CMMI Model Version 2.0. CMMI Institute, a division of ISACA, Pittsburgh, PA, 2.0 edn. (2018). Available at https://cmmiinstitute.com/cmmi/dev
5. Dittrich, Y.: What does it mean to use a method? Towards a practice theory for software engineering. Inf. Software Technol. **70**, 220–231 (2016)
6. Dittrich, Y., Michelsen, C.B., Tell, P., Lous, P., Ebdrup, A.: Exploring the evolution of software practices. In: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 493–504 (2020)
7. Dybå, T., Maiden, N.A.M., Glass, R.: The reflective software engineer: Reflective practice. IEEE Softw. **31**(4), 32–36 (2014). https://doi.org/10.1109/MS.2014.97
8. Felderer, M., Travassos, G.H.: Contemporary Empirical Methods in Software Engineering. Springer (2020)
9. Felderer, M., Travassos, G.H.: The evolution of empirical methods in software engineering. In: Contemporary Empirical Methods in Software Engineering, pp. 1–24. Springer (2020)

10. Kalinowski, M., Weber, K., Franco, N., Barroso, E., Duarte, V., Zanetti, D., Santos, G.: Results of 10 years of software process improvement in Brazil based on the mps-sw model. In: 2014 9th International Conference on the Quality of Information and Communications Technology, pp. 28–37. IEEE (2014)
11. Kitchenham, B., Dyba, T., Jorgensen, M.: Evidence-based Software Engineering. In: Proc. 26th International Conference on Software Engineering (2004)
12. Mendez Fernandez, D., Passoth, J.H.: Empirical software engineering: from discipline to interdiscipline. J. Syst. Software **148**, 170–179 (2019)
13. Popper, K.R.: The Logic of Scientific Discovery, p. 480. Basic Books, Oxford, England (1959).
14. Ralph, P.: Acm sigsoft empirical standards released. ACM SIGSOFT Software Eng. Notes **46**(1), 19–19 (2021)
15. Rombach, D.: Empirical software engineering models: Can they become the equivalent of physical laws in traditional engineering? In: Muench, J., Schmid, K. (eds.) Perspectives on the Future of Software Engineering, pp. 1–12. Springer, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37395-4_1
16. Stanley, K.W.: An Introduction to the Philosophy of Science. Cambridge University Press (2014)
17. Wohlin, C., Šmite, D., Moe, N.B.: A general theory of software engineering: Balancing human, social and organizational capitals. J. Syst. Software **109**, 229–242 (2015). https://doi.org/10.1016/j.jss.2015.08.009. http://www.sciencedirect.com/science/article/pii/S0164121215001740
18. Wong, W.E., Mittas, N., Arvanitou, E., Li, Y.: A bibliometric assessment of software engineering themes, scholars and institutions (2013–2020). J. Syst. Softw. **180**, 111029 (2021). https://doi.org/10.1016/j.jss.2021.111029

Designing a Syllabus for a Course on Empirical Software Engineering

Paris Avgeriou, Nauman Bin Ali, Marcos Kalinowski, and Daniel Mendez

Abstract Increasingly, courses on empirical software engineering research methods are being offered in higher education institutes across the world, mostly at the MSc and PhD levels. While the need for such courses is evident and in line with modern software engineering curricula, educators designing and implementing such courses have so far been reinventing the wheel; every course is designed from scratch with little to no reuse of ideas or content across the community. Due to the nature of the topic, it is rather difficult to get it right the first time when defining the learning objectives, selecting the material, compiling a reader, and, more importantly, designing relevant and appropriate practical work. This leads to substantial effort (through numerous iterations) and poses risks to the course quality.

This chapter attempts to support educators in the first and most crucial step in their course design: creating the syllabus. It does so by consolidating the collective experience of the authors as well as of members of the empirical software engineering community; the latter was mined through two working sessions and an online survey. Specifically, it offers a list of the fundamental building blocks for a syllabus, namely, course aims, course topics, and practical assignments. The course topics are also linked to the subsequent chapters of this book, so that readers can dig deeper into those chapters and get support on teaching specific research methods or cross-cutting topics. Finally, we guide educators on how to take these building

P. Avgeriou (✉)
University of Groningen, Groningen, The Netherlands
e-mail: p.avgeriou@rug.nl

N. B. Ali
Blekinge Institute of Technology, Karlskrona, Sweden
e-mail: nauman.ali@bth.se

M. Kalinowski
Pontifical Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil
e-mail: kalinowski@inf.puc-rio.br

D. Mendez
Blekinge Institute of Technology and fortiss, Karlskrona, Sweden
e-mail: daniel.mendez@bth.se

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2024

D. Mendez et al. (eds.), *Handbook on Teaching Empirical Software Engineering*,
https://doi.org/10.1007/978-3-031-71769-7_2

blocks as a starting point and consider a number of relevant aspects to design a syllabus to meet the needs of their own program, students, and curriculum.

1 Introduction

In the previous chapter (Editorial Introduction), we established the need for offering high-quality courses on empirical software engineering (ESE) to educate students in higher education institutions (mostly at MSc and PhD levels) or train practitioners working in industrial research and development. We have also elaborated on several challenges that educators face when designing such courses. These challenges are partly because the topic is still rather young, and we do not yet have dedicated textbooks or reusable syllabi to guide course designers. However, the challenges are also due to the complex nature of this topic offering ample freedom for course designers to tailor their course. This freedom is both a blessing and a curse: a blessing as it allows one to cater to the needs of individual programs, curricula, and target audiences and a curse, because such tailoring takes many iterations before an educator gets it right.

To understand how such courses are designed and run by educators, we organized two consecutive working sessions during the annual meetings of the International Software Engineering Research Network (ISERN) in 2021 and 2022. The second session was combined with an online questionnaire. The two working sessions were attended by 31 unique participants while the questionnaire was filled in by 27 respondents (15 of them overlapping with the working session participants). All aforementioned participants and respondents were affiliated (at the time) with academic institutions from Europe and North and South America.

During these working sessions and the online questionnaire, we collected a set of (a) course aims, (b) course topics, and (c) course assignments. We also had the chance to discuss with the participants about their experiences on course design. One common but striking remark from those sessions was that, by and large, every educator "reinvents the wheel." There is little to no reuse of any aspect of course design in ESE. This not only leads to substantial effort every time a course is (re)designed, but it also entails significant variation in quality.

To address this problem, this chapter aims at offering some building blocks that can provide a starting point for designing a course in ESE.[1] To be more precise, this chapter offers a collection of course aims, topics, and assignments; these are the three fundamental elements of a syllabus. Furthermore, the chapter includes guidelines on how to create a syllabus as well as a number of other pertinent aspects (e.g., ethics approval, datasets, tools, and prerequisites). The intent is that course designers, by following these guidelines, can select elements from the provided building blocks and refine them to create a syllabus to match their curriculum

[1] While this chapter refers to education in software engineering, educators can draw inspiration for similar research methods courses in related disciplines like information systems.

and program needs, student population, and, of course, personal teaching style and philosophy. By mixing and matching the elements of this chapter, we hope that course designers can reuse the collective wisdom of the ESE community, as well as their long experience in designing and teaching courses. That should effectively lead to fewer iterations in course design and better quality assurance in new or redesigned courses.

We have intentionally not provided a single, one-size-fits-all syllabus for a course in ESE, as we believe each such course is unique and requires a tailored syllabus. We thus advise educators to perform a step-wise refinement and customization of the offered building blocks to address the context of their course and of course their background. To further assist them in this task, we: (a) offer a simple set of guidelines and discuss pertinent aspects to consider in Sect. 5; and (b) link the course topics to other chapters in the book as well as external sources in Sect. 3. These links can allow course designers to delve deeper into the specific topics they want to include in their course, as those are covered in later chapters or elsewhere. For example, if a course designer has selected to teach experimentation and give an assignment to design and run an experiment, the chapter of Vegas and Juristo, entitled "A Course on Experimentation in Software Engineering: Focusing on Doing," provides excellent details, tips, and examples.

The rest of this chapter is structured as follows: Sect. 2 lists a wide range of *course aims* that are pertinent to ESE courses; Sect. 3 presents an inclusive list of *topics* that can be taught in such a course, linking them to the rest of the chapters of this book, as well as other sources; Sect. 4 discusses potential *assignments* to help students understand the theory covered in the topics. Finally, Sect. 5 proposes guidelines on how to use the aforementioned elements (course aims, topics, assignments) to create an actual syllabus, and it touches upon various miscellaneous aspects such as ethics approval or tools and datasets to be considered for the syllabus.

2 Course Aims

The following list of course aims is meant to cover a wide range of goals for an ESE course at the undergraduate or postgraduate level. As mentioned, this list is based on data collected from experts during two ISERN working sessions and from an online questionnaire. It is not meant to be an exhaustive list and it does not distinguish between undergraduate and postgraduate levels; we leave it at the discretion of the reader to reuse, refine, or add course aims according to the profile of the education program (e.g., technical university vs. classical university, faculty of natural sciences vs. faculty of computer science, program in computer science vs. program in software engineering), the level of the students, as well as the fit within the aims of the education program. These course aims can subsequently be refined into learning objectives by making them more specific and measurable.

After completing the course on ESE, students must be able to:

- **Understand philosophical stances**: List the different schools of thought in ESE research (such as positivism, interpretivism, critical theory, pragmatism, or realism), how they complement each other, and position research papers in this landscape.
- **Understand the importance and role of empirical research**: Explain the contribution of empirical research alongside other methods (such as formal methods used for rationalist arguments) in advancing the state of the art in software engineering, and recognize the importance of evidence for software practitioners.
- **Understand the basic methods and components of empirical research**: Explain the contribution of primary and secondary studies for evidence-based software engineering, define the nature and aim of typical research methods (e.g., surveys, case studies, experiments, or systematic literature reviews), and explain their core parts (such as research design, threats to validity, data collection, and analysis) as well as the difference between quantitative and qualitative data.
- **Understand how to read, interpret, and summarize research papers**: Using empirical standards and best practices for peer review, describe the typical structure of a research paper, identify the main parts (e.g., study design, data collection, and analysis), interpret both quantitative and qualitative results, and describe them to both specialists and laymen.
- **Classify and explain the problem or question under investigation**: Motivate a gap in the state of research and practice, and formulate research questions of different categories (e.g., exploratory, base-rate, causality) that are precise enough and can be answered through evidence.
- **Select a research design**: Use the formulated research questions to select a research method (or a mix of methods) that can collect the evidence to answer them.
- **Design and execute an empirical study**: Complete the study design according to specific research methods (including quantitative and qualitative data), and conduct the study following that design.
- **Perform data collection**: Execute the data collection part of the study design by developing data collection instruments or pertinent tools for several data sources to collect both quantitative and qualitative data.
- **Perform data analysis**: Analyze the collected data (both quantitative and qualitative) by visualizing (if possible, otherwise suitably representing), exploring, and subsequently reflecting upon, interpreting, and explaining the data.
- **Communicate research results**: Report the results of an empirical study by following established guidelines both in written form and orally.
- **Assemble a replication package**: Systematically collect all datasets, scripts, and other artifacts into a single package (depending also on the type of study), describe how to use it, and publish it in a permanently and publicly available online repository.

- **Develop critical thinking and a scientific attitude**: Follow a systematic process to conceptualize, analyze, synthesize, and evaluate collected data.
- **Develop theories**: Use the evidence collected from the empirical study to develop theories that advance the state of the art in software engineering.
- **Understand practical aspects of research**: Derive implications from research results for both researchers and practitioners and devise a plan to transfer knowledge to industry.
- **Understand ethical aspects of research**: Analyze the potential ethical implications of the study (e.g., treatment of human subjects, confidentiality of data), and devise a plan to address any ethical concerns, potentially to be approved by an ethics board.

3 Topics

The following is a list of topics, based on the aforementioned input from the experts and clustered into five broad themes. The list is not meant to be exhaustive, but indicative of the most *commonly* used topics in ESE courses.

1. Fundamentals of ESE—The general topics in this category are the theoretical underpinnings of ESE and help set the foundations for the research methods to be covered in a course on ESE. We have relied on the terminology proposed by Wohlin and Aurum [10] for the topics in this category.
    - Research approach/philosophical stances—A philosophical stance refers to a researcher's underlying philosophical perspective about research (or a combination of perspectives). It indicates a researcher's epistemological and ontological assumptions and impacts how researchers approach an investigation, interpret the findings, and draw conclusions. Some common philosophical stances in software engineering research [3] include (1) post-positivism, (2) interpretivism/constructivism, (3) critical theory, and (4) pragmatism.
    - Research logic (inductive, deductive, or abductive research)—As discussed in Chapter 1, empirical research follows an iterative process between theory building and empirical approaches through induction, deduction, or abduction (please see also [7]). The chapter by Klaas-Jan Stol in this book, titled "Teaching Theorizing in Software Engineering Research," discusses how findings can feed back to the theories and the overall software engineering body of knowledge.
    - Research outcome (basic or applied research)—Research outcome can be new knowledge contributions or improved solutions for practice. The chapter by Rodrigo Falcão et al. titled "Experiences in Using the V-Model as a Framework for Applied Doctoral Research" presents a framework that discusses the role and interplay of the research outcomes of basic and applied research.

- Research design—The choices made in creating a research design including research methodology, data collection, and analysis methods. These decisions and alternatives are discussed in the chapter by Jefferson Molleri and Kai Petersen titled "Teaching Research Design in Software Engineering."
- Research process—Broadly, there are three types of research processes, namely, quantitative, qualitative, and mixed research [10]. The book covers several methods that are used in these research processes. Chapters that cover quantitative research range from experimentation by Juristo and Vegas (entitled "A Course on Experimentation in Software Engineering: Focusing on Doing") to simulation by Franca et al. (entitled "Teaching Simulation as a Research Method in Empirical Software Engineering"). Chapters on qualitative data analysis by Treude (entitled "Qualitative Data Analysis in Software Engineering: Techniques and Teaching Insights") and on ethnographies by Dittrich et al. (entitled "Teaching and Learning Ethnography for Software Engineering Contexts") cover aspects relevant to qualitative research.

2. Typical research methods—The following topics cover a variety of research methods commonly employed in software engineering research. It is wise to include a range of methods in a course; although it may not be feasible to include all of them, we advise covering both quantitative and qualitative methods.

   - Action research—a strategy for investigating a problem or question within a specific real-world setting to bring about a change (see the chapter of Miroslaw Staron, titled "Teaching Action Research," as well as the chapter by Yvonne Dittrich et al. titled "Action Research with Industrial Software Engineering—An Educational Perspective")
   - Case study research—a strategy for an in-depth investigation to develop a better understanding of a phenomenon of interest in a specific real-world setting (see the chapter by Stefan Wagner titled "Teaching Case Study Research")
   - Design science—a strategy for software engineering research that emphasizes practical application through the creation and evaluation of artifacts (e.g., templates, systems, and processes) in a real-world setting (see the chapter by Oscar Pastor et al. titled "Teaching Design Science as a Method for Effective Research Development")
   - Ethnography—an empirical investigation strategy to study and understand the people and communities and cultures in their natural settings (see the Chapter by Yvonne Dittrich et al. titled "Teaching and Learning Ethnography for Software Engineering Contexts")
   - Experimentation—an investigation strategy to test hypothesis regarding causal relations in a controlled setting (see the chapter by Natalia Juristo and Sira Vegas titled "A Course on Experimentation in Software Engineering: Focusing on Doing")
   - Simulation—an investigation strategy that relies on imitating the behavior of a system in a mathematical model and then experimenting with the model

(see the chapter by Breno Bernard Nicolau de França et al. titled "Teaching Simulation as a Research Method in Empirical Software Engineering")
- Survey research—used to investigate the opinions, attitudes, and characteristics of a population by collecting data from a large sample of the population (see the chapter by Marcos Kalinowski et al. titled "Teaching Survey Research in Software Engineering")
- Repository mining—a form of archival analysis where the existing data and artifacts are analyzed for insights, patterns, and knowledge about software development (see the chapter by Zadia Codabux et al. titled "Teaching Mining Software Repositories")

3. Operationalization methods—These topics deal with executing empirical methods through data collection, analysis, and measurement.
   - Data collection methods (like questionnaires, interviews, observations, archival research, simulation, focus groups [10])—Several of these methods are discussed briefly in various chapters of the book, e.g., for simulation see Franca et al.'s chapter entitled "Teaching Simulation as a Research Method in Empirical Software Engineering", for observations see Dittrich et al.'s chapter entitled "Teaching Action Research", and for questionnaires see Kalinowski et al.'s chapter entitled "Teaching Survey Research in Software Engineering".
   - Measurement—This is a fundamental topic in ESE research, encompassing the measurement and prediction of the attributes of the software processes, products, and practitioners. The chapter by Paul Ralph et al. titled "Teaching Software Metrology: The Science of Measurement for Software Engineering" elaborates in depth on the nuances of teaching this topic.
   - Data analysis
     - Quantitative data analysis—covers descriptive and inferential statistics (e.g., hypothesis testing, Bayesian analysis, confidence interval estimation, machine learning models). Several excellent guidelines for quantitative data analysis exist, including guidelines in the context of controlled experiments [11], when comparing algorithms [1], and on Bayesian statistics [5].
     - Qualitative data analysis—covers the sense-making and analysis of qualitative data. The chapter by Christoph Treude titled "Qualitative Data Analysis in Software Engineering: Techniques and Teaching Insights" provides a thorough coverage of teaching this aspect.

4. Cross-cutting concerns—Such aspects are not specific to a particular research method, but they need to be addressed in any type of empirical study.
   - Ethics—covers, among others, research ethics, authorship, consent, and conflicting interests. In this book, where applicable, each chapter briefly discusses the relevant ethical concerns for the specific research method it covers. For a more general discussion of ethics, please see [2, 6]; interested readers can also consult [9] for teaching ethics.

- Sustainability—covers the principles and practices adhering to which can help improve sustainability[2] in research. The contribution by Birgit Penzenstadler, titled "Sustainability Competencies Informing Research Strategies for Software Engineering: A Personal Experience Report," gives a personal reflection on teaching this topic.

5. Scientific literature—This category covers the essence of what constitutes scientific literature, the peer review process, and both systematic and non-systematic literature reviews.
   - Reading and reviewing research papers—Not directly covered in the book but interested readers can consider consulting the ACM empirical standards for software engineering [8].
   - Secondary and tertiary studies—We have two related chapters here: (1) a chapter by Sebastian Baltes and Paul Ralph titled "Teaching Literature Reviews in Software Engineering Research" and (2) a chapter by Marcela Genero and Mario Piattini, titled "Teaching Systematic Literature Reviews: Strategies and Best Practices."

The course aims that an educator selects for a course in ESE should be sufficiently covered by the corresponding topics. One of the possible mappings between them is given in Table 1; this mapping can be used when checking the coverage of course aims by topics. We also use it within the proposed guidelines to design a syllabus in Sect. 5.

4 Course Assignments

Practical assignments play an important role in teaching ESE, helping to bridge the theoretical knowledge of the empirical methods with their application in practice. Assignments should provide students with hands-on experience in the various facets of empirical research, from designing experiments and conducting systematic literature reviews to building instruments for data collection or performing data analysis. By engaging in these practical activities, students not only reinforce their learning but also develop essential skills such as critical thinking, problem-solving, and effective communication. Like teaching software engineering, a course on ESE is fundamentally based on "learning by doing."

The following list provides examples of practical assignments based on the data collected from the aforementioned working sessions and questionnaire, *in alphabetical order*. The chapters elaborating on the different empirical strategies and their supplemental materials provide more specific examples and will be helpful for educators when designing practical assignments for their courses.

[2] Sustainability in this context is viewed broadly, for example, aligning with the UN Sustainable Development Goals (https://sdgs.un.org/goals).

Table 1 Mapping between course aims and topics

Course aims/topics	Research approach	Research logic	Research outcome	Ethics	Sustainability	Measurement	Research design	Research process	Typical empirical strategies	Data analysis	Scientific literature
Understand philosophical stances	x			x							
Understand the importance and role of empirical research	x	x	x	x							
Understand the basic methods and components of empirical research							x	x	x		
Classify and explain the problem under investigation	x	x		x							
Select a research design						x	x				
Design and execute an empirical study				x	x			x	x		
Communicate research results				x							x
Perform data collection				x							
Perform data analysis						x				x	
Assemble a replication package								x			
Develop critical thinking and a scientific attitude	x	x									
Develop theories			x								
Understand practical aspects of research			x								x
Understand ethical aspects of research				x			x	x		x	
Understand how to read, interpret, and summarize research papers	x	x									x

- Adapt an experiment—Students modify an existing experiment to either suit a different context or test a new hypothesis. This encourages an understanding of experimental design, variables, and replication. It also helps students to grasp the adaptability of experimental methods and the importance of contextual factors in empirical research.
- Assign group activities—Multiple groups apply different empirical methods (e.g., different systematic literature review search strategies) to investigate the same topic and compare and discuss the outcomes, fostering a comparative understanding of the methods.
- Conduct a secondary study—Students perform a secondary study (e.g., systematic review of literature, systematic mapping study) to identify, analyze, and synthesize relevant research on a specific topic. This assignment enhances skills in critical review and research synthesis.
- Analyze data—Students analyze a given dataset using statistical or qualitative methods to draw meaningful inferences. This kind of assignment teaches students how to apply statistical analysis techniques and qualitative methods to real-world data, emphasizing the interpretation and communication of results. A variant of this type of assignment is to perform a replication of an existing study by taking its dataset and reproducing the analysis; this allows to confirm (or not) the findings of that study and discuss/explain any discrepancies.
- Collect and analyze data—Students collect data and analyze it using predetermined methods, emphasizing also the linkage between data gathering methods and the quality of research findings.
- Design and execute an empirical study—Students design and carry out an empirical study following a method of their choice. For example, in the case of an experiment, students can gain hands-on experience from hypothesis formulation to data collection and analysis, strengthening their understanding of experimental designs. Alternatively, students can design and execute scoped-down versions of this assignment type by executing, for example, pilot studies, or only performing study design. It is advisable to make available a replication package with the study design, as well as any data, scripts, tools, instruments, etc.
- Develop a systematic literature review protocol—Students specify a reproducible protocol for systematic reviews to sharpen their secondary research planning skills.
- Participate in reading assignment—Students participate in reading assignments and group discussions of exemplary papers to develop their analytical skills and deepen their understanding of high-quality research.
- Present research—Students enhance their presentation and critical thinking skills by presenting scientific topics to the class, such as summaries of research papers or critical analyses of a set of papers.
- Mine repositories—Students analyze software repositories to uncover patterns and insights, getting familiar with the methodologies and tools necessary for mining extensive datasets.

- Perform a toy case study—Students design, conduct, and analyze a case study centered around a toy problem (a scaled-down, simplified enactment of a software engineering problem).
- Write a research proposal—Students outline a research proposal where they must define clear research questions, propose a robust empirical research methodology, and discuss the potential impact and contributions of their proposed research to the field of software engineering.

Clearly, some of these assignments are larger in scope and effort than others. For example, the assignment "design and execute an empirical study" encompasses many of the other types of assignments. As another example, the assignment "conduct a secondary study" is rather effort-intensive and can easily last an entire semester (if not more). Similarly, some of these assignments are better suited for individual students, while others can be optimally done in groups of students; this also depends on the level of the students (BSc, MSc, PhD, or trainee). We advise educators to take this list as an inspiration and customize each type of assignment by emphasizing particularly the timing constraints and the effort students are expected to spend on each course component (lectures, tutorials, self-studying, practical work).

Moreover, we would like to emphasize that the required effort for any kind of assignment should be realistic and match the difficulty/complexity of the assignment and the level of the student. Educators should not give assignments that are so difficult and effort-intensive that students end up cutting corners to be able to pass the course. Instead, educators should uphold high standards of quality and rigor so that students understand the importance of quality in both the research process and the outcome.

Finally, educators should look at the practical assignments as a means of achieving the course aims. If there is a mismatch between course aims and assignments, then one of them needs refinement. One of the possible mappings between them is given in Table 2; this mapping can be used when checking the achievement of course aims through assignments. We also use it within our guidelines to design a syllabus in Sect. 5. To give a more concrete feeling of what actual assignments look like, we have appended some assignments from actual courses in the online resources for this chapter (see https://zenodo.org/doi/10.5281/zenodo.11544897).

5 Putting It All Together to Create a Syllabus

The course aims, topics, and assignments described in the previous sections are a starting point, but how does one go about actually creating a syllabus for one's course? We recommend the following steps:

1. Select the course aims that fit the course, the overall curriculum, and the student population. Subsequently, derive concrete learning objectives that are SMART (specific, measurable, achievable, relevant, and time-bound). For example, the

Table 2 Mapping between course aims and assignments

Course aims/ assignments	Adapt an experiment	Assign group activities	Conduct a secondary study	Conduct data analysis	Conduct data collection and analysis	Design and execute an empirical study	Develop a systematic literature review protocol	Participate in reading assignment	Present research	Mine repositories	Perform a toy case study	Write a research proposal
Understand philosophical stances	x					x						x
Understand the importance and role of empirical research			x					x				
Understand the basic methods and components of empirical research			x									x
Classify and explain the problem under investigation			x									
Select a research design						x						
Design and execute an empirical study	x	x				x					x	
Communicate research results					x				x			
Perform data collection			x		x	x				x		
Perform data analysis			x	x		x				x		
Assemble a replication package			x									
Develop critical thinking and a scientific attitude	x	x	x				x	x				
Develop theories												
Understand practical aspects of research	x											x
Understand ethical aspects of research					x	x						x
Understand how to read, interpret, and summarize research papers			x				x	x				x

course aim "analyze data" can be refined into the learning objective "apply the constant comparison method to a collection of interview transcripts." We strongly advise educators against simply adopting a subset of the course aims but to really reflect on how to tailor them to their own context and make them specific and measurable.
2. Based on the formulated learning objectives, select the topics to be covered during the course; the mapping between course aims (from which the learning objectives are derived) and topics in Table 1 is a good starting point. Follow the references to the chapters of the rest of the book or to external sources to select teaching material, slides, and reading lists.
3. Based on the formulated learning objectives and the selected topics, select assignments that optimally fit the educator's context, also choosing appropriate datasets, tools, and assessment rubrics; the mapping between course aims (from which the learning objectives are derived) and assignments in Table 2 is a good starting point. It is important that the syllabus explains to the students why the assignments achieve the learning objectives, so that they understand the "why" and not only the "how" (e.g., why they can develop a theory from a specific dataset and why not).

The aforementioned steps touch upon a number of decisions that an educator needs to make to complete their course syllabus. These decisions include, but are not limited to, the following:

Ethics Approval The approval of the ethics board varies widely. Some universities have it as mandatory, while in other cases, it acts as an obstacle as it takes quite long, so it's difficult to fit it into the course timeline. We advise that, as a minimum, there is a discussion on research ethics and that students are made well aware of the ethical issues in their assignments and report on them. If the formal approval by the university ethics board is infeasible in the course time frame, an ad hoc ethics approval committee can be formed in the class either with the students themselves or with the educator and teaching assistants.

Reading List Each topic covered in the course needs to be accompanied by examples of good papers (for instance, by using existing Systematic Literature Reviews (SLRs) and relying on their quality assessment score to identify high-quality papers for a given topic), as well as examples of papers to be critically reviewed to identify points for improvement. Students should be encouraged to reflect and understand the strengths of the papers they read but also to have a critical attitude toward identifying elements of an empirical study that can be optimized (e.g., threats to validity). The latter should always be performed professionally and constructively. Finally, it is advisable to link the assignments with the reading list, so that students have a concrete incentive to read about a topic in depth (e.g., read an authoritative source on case study research, before performing a case study).

Datasets It is a good practice for educators to provide examples of datasets from the replication packages of specific papers or collections from Open Science repositories (e.g., https://zenodo.org/communities/seacraft/). Of course, educators

may also have their own datasets that students would need to analyze as part of their assignments. In that case, it is very important for educators to provide clear guidance on how students can analyze the dataset.

Open Science Make principles and practices of open data, open material, and open source explicit, including available and recommended license models. In combination with the datasets as discussed before, this also helps the students understand the features of a good replication package that follows sound Open Science principles. It is important to make it clear to students that different types of studies lead to different replication packages; for example, a quantitative study on open source data can generate very large and public datasets, while an ethnographic or action research study may require data redaction and/or making only parts of data available.

Tools There are many tools available for conducting the practical assignments, varying based on the topics covered within the course. These include generic tools such as reference management tools for systematic reviews, qualitative data analysis software tools, and tools for surveys. Some more specialized tools are also available (e.g., to support the automation of systematic reviews [4]). We recommend focusing on the underlying concepts and making the course as tool-agnostic as possible.

Prerequisites Make prerequisites explicit for the course, e.g., mathematics and statistics required for the students to be able to work on the assignments. It is equally important to make sure the course follows up on the background of the students. For example, the course may require that the students have a background in software engineering, e.g., by having followed courses from a software engineering curriculum and having participated in a capstone project. This of course depends also on the course level: BSc, MSc, PhD, lifelong learning, corporate training, etc. Finally, the expectations on previous research experience should be explicit, e.g., whether the students have already completed a research project such as a research internship or a BSc thesis.

Teaching Methods A course in ESE can utilize different teaching methods, including but not limited to lectures, tutorials, seminars, practical work, group or individual assignments, and lab work. While we cannot prescribe a one-size-fits-all configuration of these teaching methods, we strongly advise that the course has a very practical, hands-on nature, where knowledge and skills are acquired mostly by doing rather than by listening to lectures or studying the reading list. For example, having tutorials on how to apply the research methods in practice and labs on how to perform data collection and analysis has worked very well, in the experience of the authors. Coaching students for their assignments on an individual or group basis and with a high frequency (e.g., weekly) also ensures that students have several intermediate deadlines and get feedback early and often.

Grading Rubric Depending on the level of the course (e.g., graduate vs. undergraduate), some universities have a grading system, while others simply require attendance. In the former case, a syllabus needs to provide a grading rubric with

an explicit and detailed breakdown of how the grade is composed. Typically this rubric takes into account several components such as quality of deliverables (e.g., study design and study results, as well as Open Science artifacts), execution (e.g., choice of research method and formulating research questions, background and related work, data collection, data analysis, results, threats to validity), process (e.g., teamwork and meeting deadlines, motivation, initiative and creativity, following feedback), and presentations (e.g., content and style of presentations, answering questions).

Course Structure A week-by-week breakdown of all the teaching methods should be elaborated on the syllabus, together with all important dates and deadlines. A semester-long course has a very different structure than an intensive 5-day workshop and supports very different learning objectives. Educators need to ensure their set of learning objectives can be met within the time frame and budgeted effort.

In the online resources for this chapter, found under https://zenodo.org/doi/10.5281/zenodo.11544897, we also provide a number of course syllabi to be used as examples.

References

1. Arcuri, A., Briand, L.: A practical guide for using statistical tests to assess randomized algorithms in software engineering. In: Proceedings of the 33rd international Conference on Software Engineering, pp. 1–10 (2011)
2. Aydemir, F.B., Dalpiaz, F.: Ethics-aware software engineering. In: Chaudron, M., Crnkovic, I., Chechik, M., Harman, M. (eds.) Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings, ICSE 2018, Gothenburg, Sweden, May 27–June 03, 2018, pp. 228–229. ACM, New York (2018). https://doi.org/10.1145/3183440.3195075
3. Easterbrook, S., Singer, J., Storey, M.D., Damian, D.E.: Selecting empirical methods for software engineering research. In: Shull, F., Singer, J., Sjøberg, D.I.K. (eds.) Guide to Advanced Empirical Software Engineering, pp. 285–311. Springer, Berlin (2008). https://doi.org/10.1007/978-1-84800-044-5_11
4. Felizardo, K.R., Carver, J.C.: Automating systematic literature review. In: Contemporary Empirical Methods in Software Engineering, pp. 327–355 (2020)
5. Furia, C.A., Feldt, R., Torkar, R.: Bayesian data analysis in empirical software engineering research. IEEE Trans. Softw. Eng. **47**(9), 1786–1810 (2021). https://doi.org/10.1109/TSE.2019.2935974
6. Gotterbarn, D.: Software engineering: code of ethics and professional practice. In: Gonzalez, T.F., Diaz-Herrera, J., Tucker, A. (eds.) Computing Handbook, Third Edition: Computer Science and Software Engineering, pp. 74: 1–13. CRC Press, Boca Rato (2014)
7. Mendez Fernandez, D., Passoth, J.H.: Empirical software engineering: from discipline to interdiscipline. J. Syst. Softw. **148**, 170–179 (2019)
8. Ralph, P., Ali, N.b., Baltes, S., Bianculli, D., Diaz, J., Dittrich, Y., Ernst, N., Felderer, M., Feldt, R., Filieri, A., et al.: Empirical standards for software engineering research (2020). arXiv preprint arXiv:2010.03525
9. Towell, E.: Teaching ethics in the software engineering curriculum. In: 16th Conference on Software Engineering Education and Training (CSEE&T 2003), 20–22 March 2003, Madrid, Spain, pp. 150–157. IEEE Computer Society (2003). https://doi.org/10.1109/CSEE.2003.1191372

10. Wohlin, C., Aurum, A.: Towards a decision-making structure for selecting a research design in empirical software engineering. Empir. Softw. Eng. **20**(6), 1427–1455 (2015). https://doi.org/10.1007/s10664-014-9319-7
11. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering. Springer, Berlin (2012)

Part II
Fundamentals for Teaching Empirical Software Engineering

Teaching Theorizing in Software Engineering Research

Klaas-Jan Stol

Abstract This chapter seeks to support software engineering (SE) researchers and educators in teaching the importance of theory as well as the theorizing process. Drawing on insights from other fields, the chapter presents 12 intermediate products of theorizing and what they mean in an SE context. These intermediate products serve different roles: some are theory products to frame research studies, some are theory generators, and others are components of a theory. Whereas the SE domain doesn't have many theories of its own, these intermediate products of theorizing can be found widely. The chapter aims to help readers to recognize these intermediate products, their role, and how they can help in the theorizing process within SE research. To illustrate their utility, the chapter then applies the set of intermediate theorizing products to the software architecture research field. The chapter ends with a suggested structure for a 12-week course on theorizing in SE which can be readily adapted by educators.

1 Introduction

The ancient Greeks are widely acknowledged to have made major advances in the sciences [75]. Why might this be? Nisbett describes [75, p. 4]:

> The Greeks, far more than their contemporaries, speculated about the nature of the world they found themselves in and created models of it. They constructed these models by categorizing objects and events and generating rules about them that were sufficiently precise for systematic description and explanation. [. . .] Whereas many great contemporary civilizations, as well as the earlier Mesopotamian and Egyptian and the later Mayan civilizations, made systematic observations in all scientific domains, only the Greeks attempted to explain their observations in terms of underlying principles.

The success of the ancient Greeks thus lay in recognizing the value in creating models, categories, concepts, rules, and principles to establish an understanding of

K.-J. Stol (✉)
Lero and University College Cork, Cork, Ireland
e-mail: k.stol@ucc.ie

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2024

D. Mendez et al. (eds.), *Handbook on Teaching Empirical Software Engineering*,
https://doi.org/10.1007/978-3-031-71769-7_3

the world. How does research today fare in terms of these activities? More recently than ancient Greek civilization, Suddaby [114] lamented that a common problem with grounded theory is "a failure to 'lift' data to a conceptual level," and indeed, this is also a shortcoming within SE research [111]. Robson [89] argued that "without theory the research may be easier and quicker, but the outcome will often be of little value."

The last statement should hit a nerve among SE researchers. The SE community produces thousands of research papers each year, many of which have gone through a tedious review process and have found a willing audience among fellow researchers, and yet their value appears to be of little relevance or interest to SE professionals [11, 122]. At the same time, a lack of attention for theory and conceptualization has been observed within the SE community [57, 109]. I argue that these two observations, namely, (1) that much of SE research is not valued by practitioners and (2) SE research lacks attention for theory and conceptualization, are correlated. Perhaps, as a community, we should learn a lesson from the ancient Greeks and focus on explaining our observations beyond the immediate settings we study and on identifying underlying principles of our findings. This chapter seeks to draw attention to increase the reader's sensitivity to not only the need to theorize but also to suggest a lightweight approach to get started on this endeavor. Or, with apologies to Larry Wall,[1] "Theory development is hard, Let's go theorizing. . . . "

The SE community has made major strides in improving researchers' level of expertise in using a wide range of research methods. However, conducting an excellent case study or a carefully designed experiment does not, on its own, add to the body of knowledge within SE. Whereas the remaining chapters in this volume focus on teaching specific research methods, we also need to consider the role of theory to advance science. Perhaps the single-most important aspect of any study is its motivation, its grounding in prior work, and how this feeds back to the literature. Theorizing can help to address this, because it encourages to build on or challenge prior work, and theorizing effectively can suggest promising research directions. Theory development is a fundamental research strategy [110], but one that has not received much explicit attention. SE researchers simply aren't trained to theorize, and as a result, the publication norms within the SE community are such that nobody appears to miss it, either.

1.1 The Need for Theorizing

Why do we need theorizing? There are several reasons. The first reason to engage in theorizing is that this distinguishes science apart from consulting or journalism. Presenting some findings that remain completely disconnected to other findings

[1] Perl creator Larry Wall's 11th State of the Onion speech was entitled "Programming is Hard, Let's Go Scripting. . . " (https://www.perl.com/pub/2007/12/06/soto-11.html/).

without drawing any parallels does not help a scientific field forward. In other words, without theory, there is no cumulative tradition, no "glue" that holds studies together, nor is there any hope for parsimony: tens or hundreds of studies might find similar findings, all slightly different, without any attempt to synthesize, generalize, and unify such findings into a coherent theory. Each study stands on its own, and while it may reference other studies on a particular topic, there is no attempt to provide any link at a conceptual level that helps us understand trends, causes, and effects. Dubin distinguished between "being a good reporter" and "being a good scientist"; the former can present new and useful information to the body of knowledge, but the latter is able to find relationships between "things" [35]. This can readily be seen in many SE papers: "Related Work" sections read as obligated shopping lists of references to please reviewers, rather than an attempt to synthesize the state of knowledge relevant to the study that is presented ("good reporting," not "good science"). The common practice in SE papers of placing such a section *at the end* of a paper exacerbates this and is therefore misguided.

Second, and closely related to the argument above, is that due to a lack of theory or conceptualization, we remain stuck at the level of the idiosyncratic, the specific, without seeing the "big picture." Theory helps us to focus on what is important, rather than what is incidental. Without theory we tend to focus on the "here and now," rather than on what we can learn to serve us in the future. That is, rather than focusing on the latest trends, which at the time of writing seems to be the use of large language models (LLM) in software engineering (SE), and highlighting how useful specific features of a given version of ChatGPT might be, we ignore more general questions that consider fast-emerging technologies in general and how they can be leveraged in SE in practice. By the time a paper is published, the study may be outdated as the next version of the LLM is released! What can we learn from quickly emerging AI technology *in general*? Or more generally stated, what can be a *lasting contribution* to the field, rather than one that is outdated by the time the paper is presented at a conference?

Third, theorizing can help to explain seemingly "obvious" findings. Many papers receive review comments stating that "the findings are obvious." The first line of defense against this is to state that, without an empirical study, we cannot know whether something is true; perhaps our assumptions are not true at all! Theorizing plays a key role here because it can help to unpack a line of reasoning, to lay bare the assumptions that may be held within the literature, and to make explicit, with rigorous and sound argumentation, what the underlying process might be that underpins the findings. If the data support the hypothesis, then besides presenting the first empirical evidence for it, a paper would also make explicit why those findings make sense. If findings are counterintuitive, this by itself would also be very interesting. It is thus the unpacking of a given phenomenon or topic in detail—the theorizing—that would be the contribution.

1.2 Overview of This Chapter

This chapter draws on insights from other fields, in particular information systems (IS), to recast the discussion from "theory in SE" to "theorizing in SE." Specifically, this chapter is organized around 12 intermediate products of theorizing identified by Hassan et al. [47]. In previous work, I used the term "theory fragments" to refer to a similar idea [109]. I define "theory development" as a research strategy [110] that explicitly sets out to develop new theory. *Theorizing* is the process of engaging in theoretical development without the explicit goal of presenting a sufficiently developed theory.[2] Clearly, these activities are related; a scientific discipline needs both, but in this chapter I focus on the latter.

The goal of this chapter is twofold. First, it provides a gentle introduction to the 12 theorizing products, what they mean in general, and what they mean in a SE research context. The chapter then demonstrates how some excellent SE scholars have used these theorizing products. It is worth noting that one could fill a chapter on each of the 12 theorizing products. Many authors have presented insightful accounts on topics such as myths and metaphors; the current chapter only scratches the surface. Studying each of these is left to the reader as the proverbial exercise. Second, this chapter encourages the reader to introduce conceptual thinking into their research. In what ways can a topic of interest be elevated to a more conceptual level? What theorizing products have been used in the reader's field of interest? How can the reader incorporate these in their own research?

Learning Objectives
The learning objectives of this chapter are:

1. To understand and appreciate the role of theorizing in SE research
2. To understand and apply the 12 intermediate theorizing products in the context of SE research
3. To recognize theorizing in SE papers, by reconstructing the theoretical advances made using the 12 intermediate theorizing products

2 Theory Development in Software Engineering Research

2.1 Views on Theory in Software Engineering Research

Whereas there have been some efforts within the SE literature to pay more attention to "theory," the SE community has, by and large, ignored these. One reason for this

[2] Theories are arguably never complete, because they can always be extended in one way or another, hence the phrase "sufficiently developed."

is that theory development is difficult. Another reason is that there appears to be a mismatch between what researchers *believe* theory is and the research that they do; researchers may believe that traditional notions of theory, such as a network of constructs and relationships among them, do not apply to their research.

These are good reasons, and they have to do with the fact that nobody appears to be able to define what "theory" *is*. SE researchers are not alone in this; it appears fair to say that there are no objective criteria by which one can determine what is, and what is not, a theory.

In this chapter I argue that, in order to learn how to theorize, we need not be overly concerned with the traditional notion of theory as a collection of constructs and relationships, but that intermediate theorizing products [47] may also be sufficient to move the SE field forward. These outcomes can take many different shapes; the chapter seeks to explain the value of each of these types of products, what their added value is, and how they can contribute to a cumulative tradition in SE research. The chapter seeks to "free" readers from thinking in fixed patterns (traditional notions of "theory") and offer new ways that might be more suitable to SE.

Researchers in the SE field have varying views on theory and its role in research. This is partly due to the fact that the SE field consists of several subfields, many of which have their own dedicated conferences and sometimes journals and, consequently, each with different ideas and norms about what constitutes "research." For example, the MSR[3] community is quite distinct from the CHASE[4] community in the way the research is done. Some researchers have looked, seemingly with a sense of envy, at other academic disciplines where it is more common to have theories that guide and inform empirical studies, which in turn extend or dispute those theories. This envy has led some scholars to wonder: where is the general theory of software engineering [57].

Others do not seem to perceive this lack of theoretical focus and may consider the state of the art within their specific subfield as constituting the theory that they contribute to. If we were to ask researchers active in requirements engineering or software testing, for example, to provide an overview of their theory, we'd get models, processes, and taxonomies (cf. [13, 81]). The term "theory" is used here in a different way, yet it is not fundamentally incompatible with how it is used in the social sciences. In fact, the intermediate theorizing products presented in this chapter can be identified in those bodies of knowledge as well. Doing so is a challenging but excellent exercise for the reader.

[3] Mining Software Repositories.

[4] Cooperative and Human Aspects in Software Engineering.

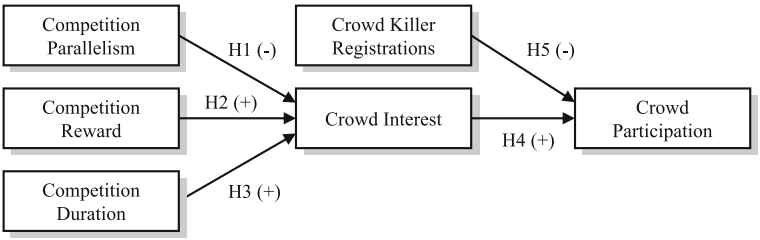


Fig. 1 Theory of crowd participation [106], an example of a variance theory

2.2 What Do Theories Look Like?

Most discussions of "theory" in SE provide tutorial-style explanations suggesting that a theory consists of theoretical concepts and the relationships between them (propositions or hypotheses) (cf. [109]). This common format is a template for what is called a variance theory: if A causes B, then A and B are the concepts, and the relationship is a causal one.[5] This means that if A goes up, then B goes up also. Figure 1 presents an example of a variance theory. Variance theories are intuitive, and they can be tested using quantitative datasets, which are relatively easy to come by, for example, by Mining Software Repositories (see Codabux et al.'s chapter in this volume) or developer surveys, notwithstanding the many challenges in both MSR and conducting surveys (see Kalinowski et al.'s chapter in this volume).

Another type of theory is called "process theory," defined by Huber and Van de Ven [53] as "An explanation of the temporal order in which a discrete set of events occurred, based on a story or historical narrative." Ralph defines a process theory as follows: "A process theory is a system of ideas intended to explain, understand (and sometimes predict) how an entity changes and develops," and Ralph discusses four "ideal types" of process theories frequently found in organizational/management research and how they apply to SE [84]. Process theories are frequently more flexible and capture a wide range of phenomena in the real world. However, these theories are not easily tested using quantitative datasets only, but rather in comparison to an alternative process theory (cf. Ralph [83]). Figure 2 presents a process theory of large-scale agile software development. This process theory explains how tensions that arise through both internal and external perturbations require an emergent response of the organization or teams involved, so as to restore a certain "balance." As such, this can be seen as a *dialectic process theory* (one of the four "ideal" types), in which "stability and change are explained by reference to the balance of power between opposing entities" [119]. In this case, the opposing

[5] Most work tends to shy away from the verb "cause" and use the more vaguely stated "is associated with" so as to not claim causality. Causality can typically not be demonstrated without conducting controlled experiments (see also Vegas and Juristo's chapter on experimentation in this volume), though there are some techniques that can provide convincing evidence for causality.

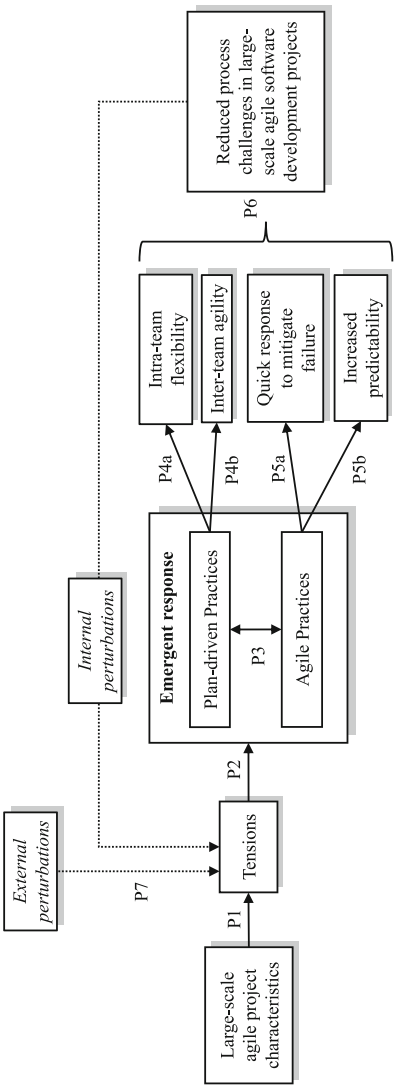


Fig. 2 Process theory of large-scale agile software development (Rolland et al. [90])

entities are the tensions that arise due to a conflict between agile processes and the large-scale nature of a software project and the emergent responses by means of adopting both plan-driven and novel agile practices that can reduce those tensions. Note that this process theory defines a number of propositions (numbered P1 to P7) and may resemble the hypotheses in the variance theory in Fig. 1, but that these are not easily testable using quantitative datasets.

While variance and process theories are perhaps the best known within SE literature, these are by no means the only forms. Other scientific disciplines and discourses follow other traditions. For example, Dittrich developed a *practice theory* on the use of software development methods [34], which:

> develops a conceptual base for understanding software development as social and epistemic practices, and defines methods as practice patterns that need to be related to, and integrated in, an existing development practice.

The format of Dittrich's presentation does not follow that of a variance or process theory, but is much more narrative. Thus, there is no clear-cut format that defines what theory is. This remains one of the major challenges in developing theory.

2.3 Challenges in Developing Theory in Software Engineering Research

As noted earlier, while much has been written over the past decades on "theory," there is no general consensus on a definition of "theory" [124], and what people accept as a theory appears to be dependent on the norms of a specific field of research. What might some other reasons be why developing theory is difficult in the SE field?

- There is a general lack of knowledge and appreciation for theorizing.
- There is a mismatch between the SE field's fast pace of development and general level of patience of researchers.
- Researchers conflate and lack understanding of terminology.

Due to a general lack of knowledge of, and appreciation for, theorizing, many researchers within the SE field simply do not pay much attention to this. Related work sections are mere shopping lists of a "who's who" in the field of interest, without a deep engagement with the actual contributions of those authors; Nobel Prize recipient Sir Peter Medawar described it as [70]:

> a section called 'previous work' in which you concede, more or less graciously, that others have dimly groped towards the fundamental truths that you are now about to expound.

In other words, the SE field lacks an effective cumulative tradition. For example, there are over 100 case studies on the use of agile methods in distributed development [33]. Through a carefully crafted introduction, many papers over-rely on

"spotting the gap," i.e., identifying the study that *hasn't* been done, without aiming to see the forest for the trees.

Another major impediment to developing theory is the time commitment that is necessary. Two related factors play a role which go against this time commitment. First, there is the SE field's pace: new technologies and trends emerge quickly. At the time of writing, the release of ChatGPT, a publicly available LLM, immediately demanded attention from many researchers leading to papers that describe this new type of technology. Second, SE researchers are not rewarded for long-term investment; it is still common for researchers to target the International Conference on Software Engineering (ICSE) with papers that are not further developed.[6] It appears that the community norm isn't such that researchers are encouraged or expected to follow up, nor to develop theory. This community norm is difficult to change. Students may not be overly concerned with this, as their funding clock is ticking. Supervisors are in a difficult position, because they prefer their students to succeed and not subject them to higher standards than would be the norm within the community, while both funding agencies and tenure evaluation committees are expecting results from principal investigators.

Finally, the terminology surrounding theory, theorizing, and theory development can be rather confusing. Most of the terms discussed in this chapter are misunderstood, misused, or conflated with other terms. For example, the terms "model" and "framework" are frequently conflated, as are the terms "concept" and "construct" [102].

The remainder of this chapter addresses these three issues. By elaborating each of the 12 intermediate products of theorizing in the next 3 sections, it addresses the last issue of terminology. It then highlights 12 intermediate products of theorizing and demonstrates how they have appeared in SE literature, drawing attention to the role these products have played in theorizing. Finally, it draws attention to the possibility of building a cumulative tradition within SE, by demonstrating how the theorizing products can contribute to a "bigger picture."

3 Theory Framing

The first set of theorizing products help in the activity of *framing* a phenomenon. The term "framing" means establishing what the topic of interest is, or positioning a research problem in the real world. Framing is the activity of finding a "hook," or problematizing prior work [5, 90] to start off the theorizing activity. This section discusses theory products to frame studies with a summary in Table 1.

[6] Without wishing to sound cynical or overly critical, I am not the first to lament this [21, 77, 78].

Table 1 Theory frames

Product	Description	Example in SE
Question	What questions are being asked depends on the discipline; a field of study can be defined by the set of questions it asks, the methods that are used to answer the questions, and other norms within the field [47]. Clearly, the questions being asked, the methods being used, and the norms within a field evolve over time	The question "What is open-source software?" has been addressed by several academic disciplines; SE research distinguishes itself from, say, economics through the use of different theories and methods to answer the question
Paradigm	A paradigm is a way of looking at the world. Within scientific inquiry, it defines how researchers ask questions and how those questions are answered. Paradigms imply a set of shared assumptions and the questions that are worth asking	The experimental paradigm in software engineering [9]; the evidence-based software engineering paradigm [60]. Software engineering as a branch of engineering [120]; software engineering as a socio-technical field [56]; software architecture as a set of design decisions [55]
Law	A law is a generalized statement that has universal applicability. Some laws are supported by both strong theory and empirical observations (e.g., Newton's universal law of gravity); others appear to be statements describing phenomena that appear to have universal application without a theoretical explanation	Conway's Law, used to frame Herbsleb and Grinter's study "Conway's Law Revisited" [49]; Lehman's Laws of Software Evolution; Hyrum's Law[7]
Framework	A framework is an analytical tool that comprises the main concepts of interests and potentially any relationships between them. Frameworks can serve many different purposes [98] and can be used to design a study or be the outcome of a study	Stol and Fitzgerald's [107] research framework to study crowdsourcing, identifying six important themes within the domain of crowdsourcing when applied to SE

3.1 Question

A starting point to frame or position a topic or phenomenon of interest is to ask questions. Hassan et al. [47] pointed out that "a discipline is defined by the set of questions it asks." Different disciplines can ask related questions pertaining to the same phenomenon, but depending on the discipline, different concepts, methods, and tools are likely to be used to answer those questions. A good example to illustrate this is the question: *What is open-source software?* OSS emerged in the 1990s as a topic of interest for several disciplines. Economists were baffled

[7] https://www.hyrumslaw.com/

by the phenomenon that developers would work *for free* to deliver high quality products [16, 67]. Information systems (IS) researchers tended to focus on OSS as a product, its qualities [80, 104], and how organizations could leverage open-source communities to build products [4, 73]. Meanwhile, SE researchers initially focused on the development process, asking questions surrounding concepts that were of particular interest to software engineers [72]. More recently, SE researchers have focused on a wide range of other aspects of OSS, such as peer review practices [87] and corporate participation in OSS projects [126].

How is a question an intermediate product of theory? Asking a question goes together with a justification, a motivation, and a perspective that is linked to a discipline, and knowledge of a field of inquiry. Doctoral students frequently spend a considerable amount of time to develop a research proposal, and it would not be uncommon for this process to take over 1 year. Identifying the question to pursue is far from trivial, because questions must come with a line of reasoning—and the lack of a clear reasoning is frequently a justified reason to reject a manuscript. The ability to ask interesting questions relies on intimate knowledge of both the state of practice and the state of the art. Asking interesting questions thus helps to articulate what is yet unknown, in a way that is relevant and important to a field of inquiry, suggesting that an answer will help the field as a whole forward. Alternatively, questions can be asked by challenging what is supposedly already known: problematizing assumptions within a given body of literature can be a fruitful approach to generate new insights [90, 95, 97].

In conclusion, a question is an intermediate product of theorizing, because it conveys not only the question itself but also an awareness and familiarity of prior literature, as well as the state of a particular field of inquiry. The question itself does not constitute a theory, but, if answered, can lead to a theory.

3.2 Paradigm

Whereas questions define a research discipline, the type of questions that are being asked within a given field depends on the *paradigm* that researchers within that discipline subscribe to.[8] A good example here is Basili's pioneering work that has emphasized the role of experimentation in SE research [10]. Basili [9] wrote:

> There is a fair amount of research being conducted in software engineering, i.e., people are building technologies, methods, tools, etc. However, unlike in other disciplines, there

[8] The term "paradigm" is considered by some as somewhat controversial due to Thomas Kuhn's use of the term in his influential book *The Structure of Scientific Revolutions* (cf. Shapere [99]). Notwithstanding, a paradigm is a way of looking at the world and comes with a set of shared assumptions. For example, in the object-oriented (OO) programming paradigm, all "things" are modeled as "objects." A paradigm is reflected in the way people think, write, and discuss about a topic of interest. The term is not limited to only refer to *research paradigms*, such as (post-) positivism, interpretivism, and critical realism.

> has been very little research in the development of models of the various components of the discipline. The modeling research that does exist has centered on the software product, specifically mathematical models of the program function.

Basili's key point here is that there had been too much focus on building tools and technologies, rather than the main components of software engineering as a discipline. He argued for a need to make *theoretical models*, not (only) mathematical (formal) models of software products, though obviously the latter is a valid way of doing research. Basili continued:

> We need research that helps establish a scientific and engineering basis for software engineering. To this end, the research methodologies required involve the need to build, analyze and evaluate models of the software processes and products as well as various aspects of the environment in which the software is being built, e.g the people, the organization, etc. It is especially important to study the interactions of these models. The goal is to develop the conceptual scientific foundations of software engineering upon which future researchers can build.

Whereas experimental SE emphasizes quantitative methods and the need to develop a conceptual scientific foundation, a next step in the evolution of empirical methods was evidence-based software engineering (EBSE), modeled after evidence-based medicine (EBM) [60].[9] The EBSE paradigm has had a major impact on the SE field. An important part of EBSE is the systematic literature review (SLR) method. This method is commonly used in medicine to collect all available evidence for a certain "intervention" (e.g., a medical treatment). Similarly, the SLR was envisaged to be a useful tool to gather all evidence regarding specific practices and tools, and indeed, countless SLRs are published every year. This is a good example of how the paradigm shapes the way we answer research questions.[10]

We can also observe a paradigm shift in how the SE community sees itself. Whereas traditionally the SE field was seen by many scholars as an engineering field [7, 21, 100, 120], there is an increasing consensus that SE is a socio-technical field [111], focusing on what has been termed behavioral software engineering [66], and that software has a major impact on society; for example, since 2015, ICSE features a Software Engineering in Society (SEIS) track. In the last decade or so, there has been increasingly attention for human aspects; more recently, studies focus on themes such as burnout, developer happiness [43], and diversity and inclusion.

In sum, a paradigm is not a theory, but helps to frame a topic of interest, to direct our focus, and to inform research questions to answer. To follow a certain paradigm, a researcher adopts a specific focus on what to study and how.

[9] Following Kitchenham et al. [60] and the emergent attention for theory in SE about a decade ago, I had suggested a new paradigm: theory-oriented software engineering [109]. It appears the paradigm has thus far not taken any root.

[10] While the idea of the SLR is a good one, unfortunately many SLRs in the field lack synthesis and theorizing that could help readers to see the forest before the trees [28].

3.3 Law

A "law" is a generalizing statement that holds true under certain conditions and which provides explanatory power [36]. Newton's law of universal gravitation in physics, for example, explains why objects fall to the ground. The general law of supply and demand in economics explains when prices of products and services go up or down (with boundaries and conditions beyond which the law may not hold, e.g., economic monopolies, price elasticity). In both cases, specific conditions may affect the extent to which these laws hold.

There are many "laws" in SE [36, 96], though what exactly makes something a law is unclear. Many of them appear to be generalized statements that are based on observations in practice, though it appears that these statements have held over time. One of these is Brooks' Law [22], which states that adding manpower to a late software project makes it later. Several researchers have taken Brooks' Law to *frame* their research [2, 52]. Hsia et al. [52], for example, conducted a system dynamics simulation study to understand the implications of Brooks' Law. The "law" suggests a certain truism, but Hsia et al.'s simulation study suggests that "there is a time line T for each project that if enough manpower is added before T, the project still can finish before the scheduled deadline," challenging the universality of Brooks' Law.

A second example of a law in SE is Conway's Law [26], which states that organizations produce system designs that mirror their communication structures. Herbsleb and Grinter [49] used Conway's Law as a starting point for their study of distributed development, where good communication, in particular informal communication, is much more challenging to achieve. Conway's Law has been used as a way to frame several other studies [14, 20].

Laws are not quite theories, but generalizations that can be used to frame a research study, to declare an initial set of theoretical expectations. The extent to which a law holds under alternative conditions can then be the focus of the study, either to confirm the law or refute it.

3.4 Framework

A framework captures the things a researcher is interested in; it describes the "map of the territory" that is of interest [71, 98].[11] A framework has been described as "a less developed form of a theory" [93]. A framework identifies the key concepts within a study; it may also capture how the concepts of interest relate to one another, but that is not necessary. Any concept that is not part of the framework is thus

[11] In this chapter I refer to theoretical or conceptual frameworks, not to be confused with software development frameworks, which are software libraries that provide functionality and may impose an architectural structure of the software that is developed. For example, Django and Flask are popular frameworks for developing Python web applications.

	Task decomposition	Coordination & communication	Planning & scheduling	Quality assurance	Knowledge management & IP	Motivation & remuneration
Customer	How do customers decompose software development tasks for crowd development?	How do customers coordinate tasks and communicate with the crowd?	How can software development tasks be planned effectively with the crowd?	How can customers ensure the quality of deliverables from the crowd?	How do customers manage project knowledge and protect their IP?	How can customers motivate the crowd, and what is an appropriate remuneration?
Crowd workers	How can crowd workers work on a single task while not knowing the rest of the system?	Do crowd workers need to coordinate among each other? How?	How do crowd workers deal with short development deadlines?	How do crowd workers test their deliverables?	What knowledge do crowd workers need to develop part of a system?	What motivates crowd workers, and what role does remuneration play?
Crowd-sourcing platform	How can platforms support both customers and crowd workers in defining tasks?	What mechanisms should platforms offer to allow effective coordination and communication?	What mechanisms should platforms offer to plan and schedule tasks?	What role can platforms play in quality assurance of deliverables?	What knowledge management solution should platforms offer to protect their clients' IP?	What guidance can platforms offer for both customers and crowd workers to come to an agreeable remuneration?

Fig. 3 Conceptual framework capturing "facets" of crowdsourcing in a software development context, as well as three different perspectives from which these facets can be considered (based on Stol and Fitzgerald [107])

not part of the "map," and thus not studied. In other words, a framework helps the researcher to focus on what is important. The conceptual framework is usually developed prior to any empirical work through a literature survey—a thorough review of literature that involves synthesis and theorizing is a study on its own [110]. As part of the empirical work, the researcher may decide to refine the conceptual framework.

An example is Stol and Fitzgerald's framework to study crowdsourcing for software development [107, 108]. The framework is developed based on an extensive survey of prior literature on crowdsourcing to develop an understanding of the process, what it entails, and who might be involved. The framework identifies six salient themes that appear to be of particular relevance to crowdsourcing in a software development context. Further, the crowdsourcing process involves different stakeholders or perspectives (see Fig. 3). This framework does not, however, capture any relationships between different concepts. Researchers can focus on one particular perspective across all six facets (one of the rows), or on one particular facet across three perspectives (one of the columns), or only on one facet from a single perspective (a single cell).

Developing the conceptual framework is a theorizing activity; there is no absolute right or wrong way of doing it, though good judgment and justification play a role. Which concepts and themes are included depends on the researcher who should justify each concept's inclusion, typically through a careful literature review. When

developing a conceptual framework, one should ensure that the map doesn't become larger than the territory; that is, researchers may be overly diligent or mechanistic and attempt to capture every little detail that could be relevant to their topic of interest. This would lead to a framework that is so complicated that it loses its strength; the point of a framework is to focus one's attention to the important elements within a domain. In my experience, frameworks that capture up to seven themes are manageable; when more themes are included, it may be useful to add some level of hierarchy to organize concepts. For example, in our study of key factors for adopting inner source, we identified no fewer than nine factors, and so we organized these into three categories [105].

In an attempt to keep the number of concepts limited, it can be tempting to combine concepts that appear to be related. It is important that this is done with utmost care, only when two concepts are so intertwined that keeping them separate is not practical. For example, several concepts within our framework for studying crowdsourcing were combined; when discussing the topic of "planning," we could not clearly and meaningfully separate this from "scheduling." Thus, if two concepts are intertwined, it is best to combine them. If there are too many distinct concepts, it is better to add some level of hierarchy by introducing categories.

A framework is not a theory, but it certainly is an intermediate product. It helps to conceptualize important elements within a domain of interest and as such to *frame* a research study. They are reusable and extendable by others and can provide a foundation for a whole stream of studies.

Exercises

1. Consider one of the following topics:
   Large language models, secure coding, diversity and inclusivity in SE
   Identify a question to ask about this topic that is relevant to the software engineering community. Why is this question relevant to SE, and why is it important? What could be a question that is not relevant to SE?
2. Two popular paradigms within SE research are evidence-based software engineering [60] and behavioral software engineering [66]. What are the assumptions underpinning each? What type of questions does each elicit? Suggest a question that is representative of each paradigm for the selected topic.
3. There are many "laws" within SE (Brooks' Law and Conway's Law as discussed above; Lehman's Laws of Evolution, etc.) [36]. Identify a law within the SE domain and how this could frame new research problems.
4. Given one of the topics listed above, perform a quick (ca. 1h), ad hoc literature review, and identify some themes that appear to be relevant to the picked topic. Create a table with these themes, and summarize for each theme why it is relevant to the topic at hand, and identify at least one question for each theme.

4 Theory Generators

A second set of intermediate products of theorizing can help to *generate* theory; these include myths, analogies, metaphors, and models [47]. They help to generate theory because they can offer an initial foundation to capture some phenomenon of interest. This section describes these (see Table 2 for a summary).

4.1 Myth

A myth is an unquestioned belief that is not supported by evidence or fact [118], or "A dramatic narrative of imagined events, usually used to explain origins or transformations of something" [50]. Hassan et al. [47] argued that myths are theory generators because "they can help uncover unquestioned assumptions within existing belief systems and theories." Myths help researchers to focus their research, to question whether the myth is true or not, and if not, pursuing an answer to the question what the truth then might be. Myths abound in SE research [58]. A

Table 2 Theory generators

Product	Description	Example in SE
Myth	A myth is an unquestioned belief that is not supported by evidence or facts [118]. Challenging a myth can help to identify a new research direction	The assumption that agile methods and plan-driven methods are mutually exclusive is a myth that Rolland et al. [90] challenged so as to generate new theoretical insights
Analogy	An analogy is a comparison of a phenomenon of interest to a phenomenon that is more familiar. The goal is to draw parallels so as to explain what something is, or how or why something works	A software system grows like a city; that is, a software system isn't a city, but it evolves *like* a city: it is in a constant state of evolution, changing some parts
Metaphor	A metaphor is a figure of speech that equates two things, thereby assigning characteristics of the one thing to the other. All metaphors are analogies, but not all analogies are metaphors	Software engineering; software development pipeline; software ecosystem; scaffolding code [22]; technical debt [64]
Model	A model is a simplified representation of a phenomenon that includes only those aspects that appear relevant and important, disregarding any details that may be idiosyncratic or incidental. A model helps to reason about the phenomenon of interest. The more detail a model has, the more realistic it is likely to be, at the cost of increased complexity	Statistical models including structural equation models; software process models, including waterfall as a simplification of the software development life cycle; computer simulations that are configured through a fixed number of parameters

classic example of this is the traditional waterfall approach to software development, described and critiqued by Royce [92]. In brief, the waterfall process describes a rational process, whereby first the requirements are gathered, followed by a system design, implementation, and then testing. While this approach appears to be completely rational in that *it makes sense* to the outsider, anybody who ever developed any software knows this is not possible. Parnas and Clements [79] discuss this mythical rational process, argue that it is an "idealization," but then provide reasons as to why a description of such a process is useful. In effect, taking the myth of the rational development process, Parnas and Clements theorized why it is a myth and continued to argue for faking it nevertheless.

Another example of this is a study by Rolland et al. [90] (briefly mentioned in Sect. 2.2), who challenged three assumptions that were widely held in relation to the scaling of agile methods. In brief, while agile methods were originally meant for small projects performed by co-located teams, it soon became clear that organizations wanted to apply agile methods at scale. A considerable body of knowledge emerged over the past decade or so, reporting on such tactics like Scrum of Scrums to scale up agile methods. In their study, Rolland et al. identified three assumptions, or myths, that appeared quite dominant throughout the literature, for example, "Agile and plan-driven methods are perceived to be mutually exclusive." This set the focus of the field study, through which they demonstrated that organizations need plan-driven practices to enable agility at scale. The analysis resulted in a series of propositions grounded in the empirical findings. Thus, the myth helped to *generate* new theory (see Fig. 2).

4.2 Analogy

An analogy can be used to explain or clarify something that is not (or less) familiar, by means of comparing to something that is known. In very simple terms, an analogy is saying X *is like* Y, whereby X is the topic at hand and Y is the analogy. While the two things differ in some ways, using an analogy means emphasizing the common aspects, finding similarities [48]. Analogies are useful to help explain a lesser known phenomenon or thing using terminology that is more familiar.

As a simple example, consider the practice of pair programming. When we say that pair programming "is like riding a tandem bicycle," we are using an analogy to convey that two people are working together to make progress on a task.[12] We're not suggesting that pair programming *is* riding a tandem bike. Analogies are closely related to metaphors, discussed below; while all metaphors are analogies, not all analogies are also metaphors. Within pair programming, we rely on the metaphors "driver" and "navigator" to convey the meaning of the two roles engaged in pair programming.

[12] Lazy back-riders notwithstanding.

Table 3 Nobody makes a comparison like this to explain what the Perl language is, because duct tape is used as a figure of speech. For the duct tape metaphor to work, it is important that people know what it is; else it is not a good metaphor

Attribute	Duct tape	Perl language
Stickiness	Ties things together	Ties different programs together
Versatility	Easy to tear off a piece to use quickly; easy to use on any surface	Easy to fit in a Perl script into any operating system environment, including practically all Unix and Linux variants, Windows, and macOS
Maintainability	May not last long	Perl scripts sometimes seen as unmaintainable due to the Perl syntax that doesn't suit everybody and which some have described as "line noise" and Perl as a "write-only language"

An analogy is a very effective literary device to explain an otherwise difficult or unknown topic. To give another example, component-based software development (CBSD) was a popular approach in the 1990s that attracted considerable attention from SE researchers. It can be explained by analogy as building a house from standardized components that are prefabricated off-site [62]. The analogy helps in understanding that components are built "off-site" (i.e., by other teams or companies), that they have some standardized interface to connect them together, and that they exhibit certain pre-defined qualities. However, the analogy may also explain why CBSD as originally envisaged has not quite become common practice and faded as a topic of research:[13] in building construction, components that don't quite fit well can be glued together with expanding foam, which is commonly used to seal gaps around cavities and is ideally suited to fill irregular shapes. In software development, however, we don't have the equivalent of expanding foam: interfaces, data formats, and protocols must be strictly adhered to; otherwise the system as a whole doesn't work (cf. Garlan et al. [39]). I would be remiss not to mention the Perl programming language, known as the "Duct Tape of the Internet" [46], but this is more a metaphor than an analogy, because we don't *compare* the Perl language to duct tape in a structured manner (see Table 3). Metaphors are discussed next.

[13] This is arguably a pessimistic view and depends on one's definition of "component." In the 1990s, standard component architectures such as Microsoft's COM (and later DCOM) and Java's JavaBeans appear not to have achieved the widespread adoption that had been expected. At the same time, if we treat open-source libraries as components, software reuse has become an incredible success. Software ecosystems found in many popular languages (including NPM (Node.js, JavaScript), R, and Rust) offer fantastic environments to build new software products quickly from these components.

4.3 Metaphor

Metaphors are similar to analogies, but whereas an analogy is stating that X is *like* Y, a metaphor is stating that X *is a* Y.[14] The difference is small and sometimes subtle, also because all metaphors are analogies, but not all analogies are metaphors. A few examples follow.

When we use the term "publication pipeline," we are using a metaphor to convey the process of producing research papers. Implied here is that a number of papers are "in production" in various states of readiness. What we don't mean is that we have an actual pipeline in our labs into which we pour research data and which produces papers as output.[15] When we say that we don't have any "bandwidth" to take on more research projects, we are using the term metaphorically to convey we are at capacity. To stick with the domain of publishing our research, we can treat our academic publications as a "conversation," whereby each publication contributes to that conversation [12]. Following this metaphor, it would only make sense to contribute to a conversation if you first listen carefully to others—that is expected in academic papers through a related work section. Thus, this metaphor helps us to theorize about how academic publishing works or should work. As discussed in Sect. 2.3, the SE literature does not much resemble a conversation.

Metaphors are extensively used in the SE field [59]. One that has received considerable attention over a time span of more than two decades is the term "technical debt." The metaphor was first used by Cunningham [29] to describe the danger of shipping "not quite right" code and the trade-off that is made in order to be able to ship a product fast:

> Shipping first time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite. Objects make the cost of this transaction tolerable. The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt. Entire engineering organizations can be brought to a standstill under the debt load of an unconsolidated implementation, object-oriented or otherwise.

Kruchten [64] explicitly recognized the role of this metaphor as a theory generator in their editorial *"Technical Debt: From Metaphor to Theory and Practice."* Lamenting that any issues in software development could be labeled some sort of "debt," Kruchten et al. argued that:

> We need a better definition of what constitutes technical debt and some perspective or viewpoints that let us reason across a wide range of technical debt. In short, we need a theoretical foundation.

[14] This is a simplification, and does not necessarily imply that the decision whether something is a metaphor or analogy depends on the presence of the words "is like."

[15] Notwithstanding the recent developments of large language models.

4.4 Model

Similar to the term "framework," the term "model" also has different meanings in a SE research context. Whittle et al. [123] succinctly describe that "in software engineering, a model is an abstraction of a running system." This approach to software engineering is better known as model-driven engineering (MDE).[16] Renowned theorist Robert Dubin equated the term "model" to mean "theory," though acknowledged that not everybody agrees with this view [35]. In this chapter, following Hassan et al. [47], I use the term "model" not to refer to a theory, but as an intermediate product of theorizing.

Consistent with Whittle et al.'s description, the term "model" usually implies that it represents an abstraction from something in the real world, and in the context of this chapter, that is how I use the term model. Abstraction implies simplification and the leaving out of details. Curran [30], discussing statistical modeling, describes the importance of simplification:

> If I build a model 787 in my backyard, and then fly 382 people to Charles de Gaulle, it's not a model 787, it is a freakin' 787. It is the simplification that is the point.

Statistical models are common in SE research, in particular multiple regression analyses. Less popular, though more powerful and flexible, are structural equation models (SEM) which are statistical models that can be easily visualized as box-arrow diagrams, which are a common way to represent variance theories (see Sect. 2.2). Statistical models have an associated model *fit*; in multiple regression this is R^2, indicating the variance explained by the fitted regression line, and in SEM there are a whole range of fit indicators [112]. These measures of fit indicate how well the theorized model "fits" the data sample and acknowledge that models are, in the end, simplifications that leave out details. In practice this means not only that variables are left out of a model (thus reducing the R^2) but also the fact that a quantitative analysis cannot capture the idiosyncratic and the specific details that could provide further insight.

A well-known statistical model in SE is Boehm's COCOMO cost estimation model [15]. But not all models are statistical. Redwine and Riddle's [86] technology maturity model (even though they didn't call it a "model") describes how technologies mature and get adopted over time, which can help to predict and set expectations. Models are used extensively in the literature on software processes [91], which describe processes, though models could describe other things as well.

[16] One particular subset of MDE is Model Driven Development, which seeks to generate code based on a model. In MDD, the model must be exactly right so that correct code can be generated—one could say that the model can be executed; generating and compiling the code are just intermediate steps. Because details now matter and must be included for the model to be executable, it is no longer an abstraction but just a different way for a programmer to express their intention (instead of code). Therefore, the term 'model' in MDD diverges from how the term is used in this chapter. In contrast, in research we create models to simplify the topic of interest so as to make sense of it, to reason about it, and to analyze it.

The waterfall approach described earlier (see Sect. 4.1), which was documented and criticized by Royce [92], is a model of the life cycle of software development. The utility of such models like those described by Redwine and Riddle [86] and the waterfall model can be assessed by looking at how closely they represent reality—how well these models "fit" with reality. As it turns out, the waterfall model is a rather poor fit with reality, in that most software development doesn't actually follow such a linear process.[17]

Models provide the researcher to describe something in a systematic way so as to communicate the important characteristics, leaving out the idiosyncratic issues from the real world. As an abstract representation, researchers can reason about it, perform analyses, and even run computer simulations which themselves rely on simplifications [110]. Simulations are discussed in more detail in De França's chapter in this volume.

Exercises

1. Consider your own research topic. Attempt to find a paper that presents a "model" in relation to the topic and also a paper that presents a "framework" for the same topic. How are they used? How do they differ?
2. Discuss the difference between an analogy and a metaphor. How do they differ?
3. Identify a myth in the software engineering literature, and find one or two papers that discuss or investigate it. How could this myth be used as a theory generator?
4. Identify a "law" in the software engineering literature, and find one or two papers that discuss or investigate it (cf. [36, 96]). Well-known examples are Brooks' Law, Conway's Law, and Hyrum's Law. How does the law help to frame research?

5 Components of Theory

The third set of theorizing products are actual components, or elements, of theory. These include the things we wish to reason about or measure (concepts, constructs) and the relationships between those things (statements, hypotheses). These components are commonly explained in a tutorial on "theory" [109]. Because the terms concept and construct are so closely related, in defining one I make reference to the other. Table 4 presents a summary.

[17] Software development in safety-critical domains, such as the aerospace domain, automotive, and medical devices, is subject to regulatory guidelines or standards that prescribe the use of plan-driven methods.

Table 4 Theory components

Product	Description	Example in SE
Concept	A name given to an idea or set of ideas so that it can be reasoned about	Developer experience (DevEx) first proposed as a concept by Fagerholm and Münch [37] and further studied by others [76]
Construct	A proxy that is an empirical stand-in for a theoretical concept [88]; these can be modeled as a common factor (when using covariance-based SEM), a composite (when using PLS-SEM), or other forms such as sum scores, averages, etc. In SE, it is not uncommon to use singular observed variables as a proxy for a concept	Trust [112], social capital [113] [82]
Statement	A description relating to one or more concepts that is derived from logical reasoning, prior theoretical insights or literature, or empirical evidence. Includes propositions, typologies, and taxonomies	Statements include propositions (e.g. Crowston's propositions on work practices in open source software [27]), and typologies and taxonomies [84] (e.g. Lehman's SPE classification [65])
Hypothesis	A statement that has been operationalized so that it can be empirically tested	Many papers in the top conference in SE feature hypotheses; specific examples can be found in [113], e.g., the more a developer trusts other members in the inner source community, the higher that developer's job satisfaction

5.1 Concept

Concepts are "ideas that receive names" [47, 117], and as such, they are the building blocks to develop theories. A research discipline deliberates the things that scholars believe are important, who then go on to name them. They have been described as the "product of the imagination that can be conveyed to others only by means of language" [54].

The term "concept" is frequently conflated with the term "construct" (discussed next in Sect. 5.2); however, it can be useful to distinguish these as different things. Concepts represent sets of ideas, which are labeled to discuss them or theorize about them. Constructs, in contrast, are used to bring concepts to life, to operationalize them through some form of measurement. Hassan et al. [47] summarize that "constructs are attempts to make concepts less 'abstract' even though constructs are abstractions of concepts."

Not all concepts are measurable, and therefore it is not always possible to define meaningful constructs for a concept. A concept can be any "idea that receives a name," which also includes "software development process" or "software product," and yet we would not seek to measure these concepts. Instead, we would measure

their *properties* [35, p. 40]; for example, rather than measuring the software product, we'd seek to measure its various qualities. In other words, concepts can be much more generic ideas. Constructs are discussed in more detail below (Sect. 5.2); that section also describes one historical reason as to why the terms concept and construct may be conflated or even considered equivalent by some.

Of all intermediate theorizing products, concepts are perhaps the most straightforward: anything that has been labeled as something worthy of study. Some examples now follow. At the time of writing, a current topic attracting attention within SE is "developer experience" (DevEx) [37]. The DevEx concept was first defined by Fagerholm and Münch [37] as "how developers feel about, think about, and value their work." However, the question then arises as to how to operationalize this definition, and this would require us to develop one, or perhaps more *constructs*.

Another example of a concept that originated in the social sciences is the concept *social capital*, which has also been used in SE research [82, 113]. Social capital has been defined as "the goodwill that is engendered by the fabric of social relations and that can be mobilized to facilitate action" [3]. Nahapiet and Ghoshal [74] argued that social capital has three dimensions: the structural dimension, which includes network ties, the relational dimension, including trust and norms, and the cognitive dimension, which includes shared language. Social capital, network ties, trust, norms, and shared language are concepts, because they have a place in theoretical reasoning. We can define, teach, and reason about these concepts. To measure these concepts quantitatively, however, we must define constructs, discussed next.

5.2 Construct

Constructs are, in short, mechanisms to allow measurement (observation) of concepts. Ironically, "construct" is the term frequently used to refer to variables that cannot be observed directly. Whereas concepts are the entities that we theorize about, the term construct becomes relevant when we attempt to measure those constructs. The terms construct and concept are frequently conflated; some scholars appear to reject any distinction [88]. Suddaby [115] acknowledges that the term "construct" comes with "connotations of hypothesis testing and operationalization" and that for researchers following a different research paradigm (see footnote 8 in Sect. 3.2), "*concept* might be a more acceptable value-neutral term." Whether one wishes to draw a distinction or not between "construct" and "concept," it is important to achieve "construct clarity" (or conceptual clarity), i.e., clearly defined constructs (or concepts) [115].

When we adopt a quantitative research strategy, it is useful to draw the distinction between "construct" and "concept"; to clarify why, I make a brief, though technical, diversion. I do this by drawing on Rigdon's explanation of what Bagozzi and Philips called the "Holistic Construal" [6, 88].

Figure 4a presents a simplified version of the Holistic Construal Framework (adapted from Rigdon [88]), which was first proposed in 1982, at a time when

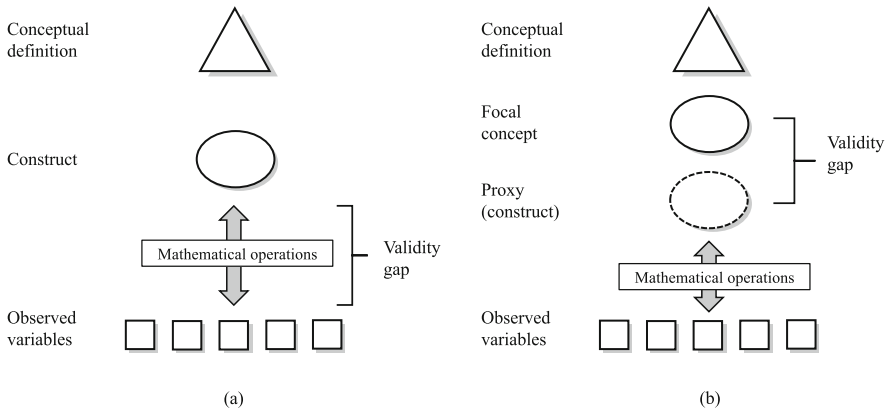


Fig. 4 (**a**) Simplified representation of the Holistic Construal Framework [6], adapted from Rigdon [88]. (**b**) Concept Proxy Framework, adapted from Rigdon [88], that explicitly acknowledges a distinction between concept and construct (proxy)

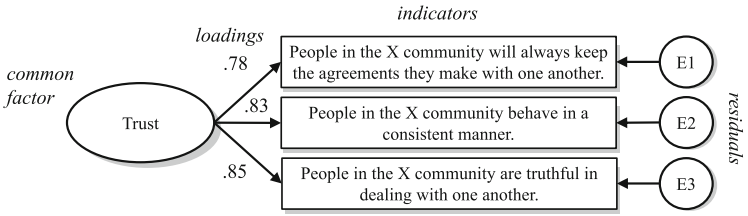


Fig. 5 Trust as a reflectively measured common factor: instead of probing respondents on "trust," which could involve misinterpretation and thus measurement error, we ask respondents to rate a set of more specific, unambiguous indicators, each of which has a certain loading on the common factor, indicating how much of its variance is explained by the common factor. Residuals explain the remaining variance in the indicators. X is a placeholder for the name of a community, not the social media platform formerly known as Twitter

statistical techniques were not yet as well developed as today. The goal of the framework was to link together *theory formulation* at the level of theoretical concepts and *theory testing* through constructs that represent those concepts.

Research involves theoretical concepts; in behavioral software engineering, such concepts include job satisfaction, trust, personality traits, and a wide range of emotions. These concepts can be clearly defined to convey to readers what the researcher means, to achieve conceptual clarity [115]. To test a theory involving these concepts, one has to *measure* these; to that end, a researcher would define *constructs*, which are created through some mathematical operation from a set of observed (empirical) variables. At the time the Holistic Construal Framework was first proposed, the dominant way to represent constructs was as reflectively measured *common factors* (see also the chapter by Ralph et al., this volume). Figure 5 shows an example for the construct "trust" (as used in Stol et al. [113])

using standard notation in SEM: ovals represent latent variables, and empirical indicators are represented by rectangles. Reflective measurement refers to the idea that a theoretical, latent variable (e.g., trust) cannot be measured directly, but only *inferred* through a set of indicators. Change in those indicators is *caused* by change in the latent variable, i.e., change in the latent variable is *reflected* in the indicators. The relationship between a common factor and its indicators is expressed as a *loading*, which indicates how much of the variance in the indicator is explained ("caused") by the common factor. Normally, not all variance in the indicators is explained by the common factor; the remainder is called the *residual*, and this accounts for measurement error.

The apparent "natural fit" between theoretical concepts on the one hand and their measurement through constructs represented by common factors contributed to a "blurring" of the terms concept and construct [88]. The lack of distinction has practical implications, because this implies that constructs *are equal to* the concepts they represent. Under this assumption, "construct validity" is then the extent to which the construct is properly constructed from observed variables.

However, Rigdon [88] has argued that the implications of equating concept with construct are untenable. Doing so would equate a concept with a common factor, the error terms on the indicators would perfectly capture measurement error, and that somehow, whatever conceptual definition the researcher had in mind, was *exactly* captured by the common factor (i.e., the construct). Rigdon argued that the common factor is merely one approximation of whatever focal concept a researcher may have in mind and that other representations are possible. He proposed the Concept Proxy Framework as a modified version of the Holistic Construal Framework (Fig. 4b), which explicitly acknowledges a distinction between the focal concept as "idealized and out of reach" and a *proxy* that is an "empirical stand-in for the theoretical concept" [88]. Like the construct in the Holistic Construal, the proxy can be created through a mathematical operation. From this perspective, the proxy may be modeled as a common factor or as something else, for example, a composite (a weighted combination of indicators) such as those in partial least squares (PLS)-SEM [88, 94]. The question of validity then shifts to the extent to which the proxy is sufficiently similar to the theoretical concept that it purports to represent.

In sum, the distinction between the terms concept and construct is useful, because in quantitative studies we must operationalize the concepts we study; we do so by means of constructs. Whether the construct is statistically modeled as a common factor or as a composite depends on many factors beyond the scope of this chapter, but whichever mathematical operation is selected, it is important to draw a distinction between the concept as something that is clearly defined (and accept that it cannot be perfectly measured) and the construct that is the "empirical stand-in" to represent that concept.

5.3 Statement

Statements are the fundamental building blocks of theorizing [47]. In this context, the term statement does not refer to just any sequence of grammatically ordered words [47]. Instead, it should be seen as an expression that makes sense within a given scientific discipline, i.e., it is relevant to what Hassan et al. [47], citing Foucault [38], refer to as scientific communication that makes up a *discourse*. For example, the statement "Continuous delivery improves software quality" makes only sense within the context of SE research.[18] Statements are claims that may or may not be presented with evidence. It is up to the researcher to (a) identify or develop the statement and then (b) offer evidence in support of that statement.

While concepts (Sect. 5.1) and constructs (Sect. 5.2) are building blocks that represent phenomena and things of interest, statements (and also hypotheses, discussed below) can be made to discuss those concepts and constructs in relation to one another.

A special type of statement is a proposition, which can be described as a "truth statement," a "prediction" [35], or assertion about concepts [47]. Hassan et al. [47] distinguish different types of propositions, depending on the type of assertion. An *existential* proposition is a nonrelational proposition that simply declares the existence of a concept. This also includes typologies and taxonomies [45, 47, 84]. A *definitional* proposition goes beyond claiming the mere existence of a concept, and unpacks the concept, providing a definition and characterization. A good example of this is work by De Boer et al. [31, 32]. They observed that, while many researchers discussed architectural knowledge, it remained "a fuzzy concept." Thus, while there was general agreement on the *existence* of the concept "architectural knowledge," they attempt to unpack *what architectural knowledge is* by drawing on different definitions and identifying its constituent elements.

Propositions should be made based on "logical links" rather than just any unjustified claim [47, 109]. This is not to say that it's not possible to present a proposition that challenges the current state of knowledge (or assumptions held within a field); rather, such a challenge should come with a *rationale*, a line of argument, a reasoning. Thus, the researcher should offer a reasonable derivation that leads up to the proposition. At the same time, there is little point in offering trivial propositions that have already been demonstrated by others, or in other fields. While any number of propositions can be proposed, Dubin suggests to aim for parsimony by focusing on *strategic propositions* [35, p. 168]; that is, ignore the trivial propositions that do not add much insight, and focus on those propositions "where something notable is happening" [35, p. 169].

[18] Note this is an example statement, not necessarily a universal fact.

5.4 *Hypothesis*

Propositions (discussed above) are statements at the conceptual or theoretical level, but in order to test them, concepts must be operationalized using empirical indicators (and/or constructs; see Sect. 5.2) [47, 109]. The result is a hypothesis. When operationalizing a proposition, it is possible that multiple hypotheses are necessary; for example, when hypothesizing about program size, one may need to use multiple metrics to operationalize this concept. These could include lines of code, number of statements, number of classes (when using object-oriented languages), or memory footprint.

Sometimes the term "hypothesis" is used in a more casual way, as if it were offered at a cocktail party. For example, closely related to Conway's Law (discussed earlier) is the "mirroring hypothesis," which can be stated as follows: "Loosely-coupled organizations will tend to develop products with more modular architectures than tightly-coupled organizations" [25, 69]. The form of this statement is more like a proposition than a hypothesis, because it only references concepts such as "loosely-coupled organizations" and "modular architectures." To actually test this hypothesis, we'd need empirical indicators (constructs); how would one measure the modularity of an architecture? For example, as part of their study, MacCormack et al. [69] operationalize a dependency between modules as a function call; if a module A calls a function in module B, then A depends on B.

Exercises

1. Discuss the difference between a construct and a concept. How do they differ?
2. Discuss the difference between a statement and a hypothesis. How do they differ?
3. Consider a topic (phenomenon) of interest, for example, your current research topic. What are some of the key concepts that appear to be important that have received attention in prior work on this topic? Likewise, are there any defined constructs?
4. Continue with the same topic; what are some key relationships that appear to be important? What are some well-known assertions, propositions, or hypotheses that appear in the relevant literature?

6 Theorizing Products in Action: The Case of Software Architecture

The theorizing products presented above are drawn from information systems scholars [47]. One could question, or even object, that such a theoretical focus is not

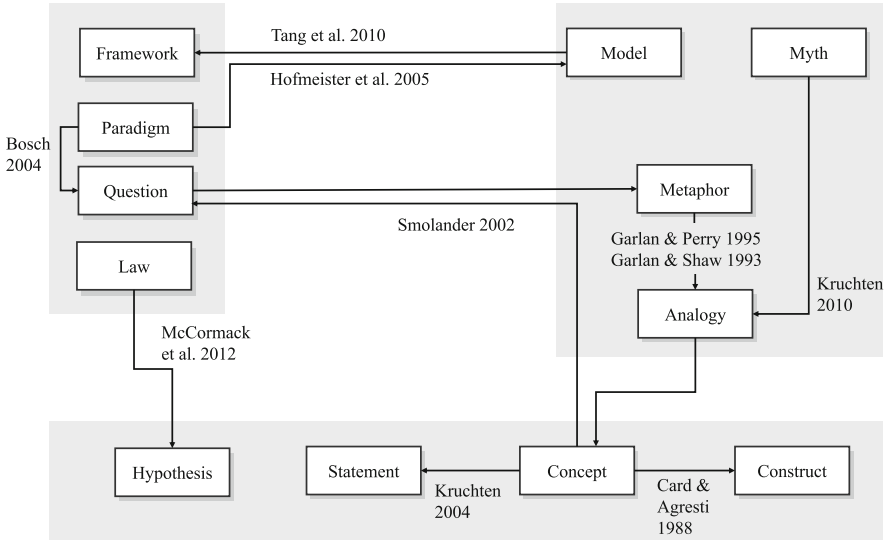


Fig. 6 Example map of products of theorizing in the study of software architecture. This map is only one illustrative example of the author's (possibly naive) interpretation of how studies relate to one another. The author acknowledges this map is incomplete. Further, in creating a map, any theorizing product can be linked to any other. Creating a more complete map can be a useful yet time-consuming way to learn about a research area

applicable to a practical field like SE, a field that some perceive as an engineering field. To highlight the value of these theorizing products, I now demonstrate how theorizing takes place within the software architecture field, an area of research that is firmly rooted within the SE domain. I adopt Hassan et al.'s approach [47] to illustrate how studies move the field forward by leveraging one theorizing product, to move to another theorizing product (Fig. 6). The presentation below should not be construed as a complete and necessarily accurate presentation of how theorizing *actually* took place. Further, while the various studies all relate to the software architecture field, the studies used to illustrate the method link to different topics, including software architecture as a concept and architectural knowledge. Reflecting on how an area of research evolved using the 12 theorizing products as a lens can help to increase awareness of theorizing so that future researchers can also learn how to reflect on a field.

6.1 From Law to Hypothesis

Among the early literature concerned with the structure and decomposition of software, years before the term software architecture was coined, is an essay by Conway who suggested that the structure of a system will follow the communication

structures that exist within the organization that produces that system [26]. This has been labeled Conway's Law in the context of software systems, but similar observations were made by others, and this has become more generally known as the "mirroring hypothesis." MacCormack et al. phrased this as follows [69]: "Loosely-coupled organizations will tend to develop products with more modular architectures than tightly-coupled organizations."

6.2 From Metaphor to Analogy to Concept

The software architecture subfield has deep roots dating back to the very early days of software engineering but started gaining traction in earnest in the mid- to late 1980s. Garlan and Perry [40] noted that one of the trends that prompted the attention for software architecture:

> For example, the box and line diagrams and explanatory prose that typically accompany a high-level system description often refer to such organizations as a "pipeline," a "blackboard-oriented design," or a "client-server system."

Pipeline, blackboard, and client-server are all metaphors that convey a particular architectural style, the overall structure of a system. There was an increasing recognition that there is such a thing as software design at the system level, rather than at the detailed level. Up to that point, detailed design would have been captured using dataflow diagrams [125], for example. One of the fundamental papers introduced software architecture as follows [41]:

> As the size and complexity of software systems increases, the design problem goes beyond the algorithms and data structures of the computation: designing and specifying the overall system structure emerges as a new kind of problem. This is the software architecture level of design.

Thus, an analogy was drawn to the high-level structure of a system, as suggested by the metaphors representing the architectural styles, similar to the idea that one can observe different architectural styles in buildings throughout the ages, and across the world. The term "architecture" is a metaphor[19] for high-level design of a software system and quickly found its way through into software companies, for example, in the naming of the role software architect.[20] By now, the term

[19] "Architecture" is the metaphor, not the term "software architecture," because prior to the coining of the term, it didn't exist. The same applies to "technical debt": debt is the metaphor.

[20] The analogy between building systems and construction has not gone without critique [8]. I agree that the metaphor is not perfect: software architects would normally have a strong development background, whereas builders don't usually become architects. Another point where the analogy with construction breaks down is the fact that architects in construction do, in fact, focus on details, aesthetics, and elegance. For example, architects would design "hidden" gutters and downpipes as these essential elements would take away from the aesthetics of a building when visible.

"architecture" (in the context of software systems) has become so common it is a "dead" metaphor [42].

6.3 Deriving Constructs from Concept

Card and Agresti [24] suggest that "architectural design complexity derives from two sources: structural (or intermodule) complexity and local (or intramodule) complexity." In effect, Card and Agresti derived two constructs from the concept of architectural design complexity, namely, the structure of the system as a whole and the complexity of individual components; each of these constructs is measured using specific metrics. While the definition of constructs is not a theory, defining constructs is essential to make any "theory" (in the broadest sense of the word) of software architecture testable.

6.4 Exploring Concept by Metaphor

While software architecture had emerged as a concept, mostly as a level of design that went beyond algorithms and data structures, the question "What is software architecture?" did not have a clear answer. Whereas there have been numerous working definitions, Smolander [103] observed in 2002 that the meaning of the concept remained ambiguous and that academic and industry perspectives diverged [17]. Around that time, the Software Engineering Institute (SEI) maintained a list of close to a hundred different definitions of software architecture. Smolander investigated how software companies perceived software architecture and identified four metaphors: architecture as a blueprint, architecture as literature, architecture as language, and architecture as decision, representing the different roles that software architecture plays in practice. The last definition, architecture as decision, foreboded a paradigm shift, discussed next.

6.5 Shift in Paradigm Leads to Shift in Questions

The software architecture as a topic of interest has emerged and evolved rapidly over the past few decades. Initially the focus was on technical artifacts, considering architecture as a collection of components and connectors that exhibited a certain structure and behavior (e.g., performance, reliability, and other quality attributes) [23]. Researchers studied how to capture this using different views, using architecture design methods and architecture description languages. Bosch [18], among others, pointed out that this view on software architecture lacks attention for capturing design decisions, and consequently the knowledge about "what and

how" is not captured in the software architecture product. This focus on design decisions heralded a paradigm shift; rather than focusing on software architecture as the end product, the focus was turned to the architectural design process, including capturing design decisions and managing architectural knowledge [23].

As a result, researchers started asking different types of questions. Instead of asking how we can capture different architectural styles or patterns in UML, for example, researchers now were focusing on defining architectural knowledge [32] and how it could be managed.

This second paradigm of software architecture is not a theory, but helped to frame the research done in this field, the questions that researchers ask, and ultimately the theorizing products that come out of that inquiry.

6.6 From Concept to Statement

The "design decision" concept gained a lot of traction within the SA domain as it opened up a new view on what should be captured. Numerous templates have been proposed to capture various attributes of design decisions. Several researchers then started to study the concept of design decisions in more detail by not only proposing templates that capture the various attributes of a design decision but also taxonomies and ontologies. For example, Kruchten [61] presented an ontology of design decisions that defines different types of design decisions and different types of relationships between those decisions. In so doing, Kruchten theorized convincingly, using examples from an air traffic control system, that there are "existence" decisions (a system will have a certain element or artifact), "property" decisions (a system will have a certain enduring trait or quality), and "executive" decisions (decisions about the process, technology, or tools to be used). While the ontology is not a theory, it did help advance the SA field.[21]

6.7 From Model to Framework

A variety of different architecture design methods emerged over time, mostly independently as they were created by different organizations in practice. Observing that these methods vary in terminology and focus, Hofmeister et al. [51] distilled a general model of architecture design. This model very much aligned with the first paradigm of software architecture research, focusing on the software architecture as an end product. Tang et al. [116] extended this model to capture the full life cycle of architecture, which includes architecture implementation and maintenance,

[21] A crude measure of impact is that Kruchten's paper was cited 420 times at the time of writing.

in recognition of the fact that the architectural knowledge created in the early phases is used later, including when a system has been deployed and must be maintained.

Whereas the extended model captures the full life cycle of architecture design during which architectural knowledge (AK) can be created and used, Tang et al. [116] then turned their focus on the activities that produce and consume AK. In so doing, they identify 12 different types of activities through which producers and consumers generate and use AK.

6.8 Debunking Myth by Analogy

In the earlier days during the rise of agile methods in industry, there was an apparent conflict between elements of the Agile Manifesto and software architecture as a "Big Up-Front Design" [63]. This myth is rooted in a false dichotomy between agile methods and architecture [1], which could be the result from a superficial reading of the Agile Manifesto. Kruchten [63] approached to resolve this mythical conflict by drawing an analogy to the ethno-sociological concept of culture and intercultural conflicts, emphasizing that the conflict can be reconciled by improving mutual understanding and foregoing "a shallow caricatural view of the 'other culture.' "

7 Teaching Theorizing: A Practical Guide

This chapter presents a wide range of theoretical concepts (pun intended) and, in so doing, scratches only the surface of each of the 12 intermediate theorizing products. Table 5 offers a practical guide to teaching this material; it should be noted that a 12-week course could focus on any of the 12 topics, or just a subset of these, informed by the lecturer's preferences. It is assumed there are 2 contact hours per week, for a total of 24 hours for the whole semester. This might seem overindulgent given there are so many other topics to teach, but this is, in the end, complicated material that requires considerable engagement lest the effort become meaningless. When faced with less time, rather than attempting to include too much material, I would suggest to focus on a select few themes and discuss these in depth.

Whatever the focus and scope are of one or a series of lectures or tutorials, it is important (a) to read one or two papers that set the stage (Table 5 offers useful starting points, but these recommended readings can easily be swapped for others) and (b) to apply the material from the paper in one way or another. An effective question is "Given the reading assignment, what elements could be applied to *my* situation or research interest?" or "What would this look like when applied to *my*

Table 5 Sample program for a 12-week course to teach theorizing in SE. Listed activities are examples only to get started, but lecturers are recommended to add or replace contents as they see fit

Week	Topic	Readings, questions, activities
Week 1	Introduction; metaphor and analogy	Read Weick on theorizing [121]. Find a few papers in your field of interest that use metaphors or analogies. Which analogies are also metaphors, and why? Which analogies are not, and why?
Week 2	Questions	Read Sandberg and Alvesson [97]. Identify a topic of interest, and identify two scientific disciplines (or sub-disciplines) that study it, for example, SE, IS, CSCW/CHI, management, economics, psychology, etc. Attempt to identify questions asked in each discipline on this topic
Week 3	Concepts	Identify a concept that has attracted interest in the past several years; examples include productivity and developer experience. What definitions are provided, if any?
Week 4	Constructs	Read Suddaby [115], Graziotin et al. [44], Sjøberg and Bergersen [101], and Ralph and Tempero [85]. Advanced reading is Rigdon's work on measurement, e.g., [88]. Discuss the differences between concepts and constructs; discuss construct validity; consider some concept within SE, and find an associated construct to measure it; is the construct a good representative of the concept?
Week 5	Frameworks	Read Leshem and Trafford [68] and Schwarz et al. [98]. Given a substantive research topic or question, attempt to create a (initial) framework to help answer this question
Week 6	Laws	Read Conway's short article [26], and discuss the strength of evidence that Conway proposed; is the term "law" appropriate? Did later studies provide support? Identify some other "law" in SE: attempt to find the origin of the law and papers that have cited the law to frame a study
Week 7	Myths	Identify a myth in SE, for example, see Bossavit [19]. Attempt to identify the origin and a paper that refutes the myth
Week 8	Models	Consider the model of architectural knowledge by Hofmeister et al. [51] and the extension by Tang et al. [116]; attempt to identify a model within a different field. Discuss the difference between a framework and a model
Week 9	Paradigms	Read Basili [10] and Kitchenham et al. [60], and compare the two viewpoints; how do Basili and Kitchenham et al. converge or diverge? Consider a research topic/field, and attempt to identify trends over the past 20–30 years, e.g., sampling papers every 4–5 years; what appear to be assumptions within the field? Are there any "shifts" in focus when comparing earlier versus recent research? Is there a shift in methods used? Attempt to compare early vs. current papers using a table. Can you speak of a paradigm shift?

(continued)

Table 5 (continued)

Week	Topic	Readings, Questions, Activities
Week 10	Statements and hypotheses	Read Crowston et al. [27]. How did they develop the propositions from prior literature? Are these propositions "strategic" and "interesting"? If so, why? If not, why not? Read Ralph for more guidance on taxonomies [84]
Week 11–12	Theorizing map; wrap-up	Build a theorizing map (see Sect.. 6, Fig. 6) on a topic of choice, ideally linked to the same concepts. This is time-consuming and challenging but is an excellent activity to explore historical developments within a field of interest

research topic?" I have attempted to do this for the software architecture field in Sect. 6.[22]

All 12 topics are "rabbit holes,"[23] i.e., it is easy to get lost. This is both the magic but also the curse[24] of doing metascience; it is magical because there is so much more to learn, but it is a curse because there doesn't appear to be a final answer. Welcome to theorizing!

Acknowledgments This work is supported with funding by Science Foundation grant 13/RC/2094-P2 to Lero, the SFI Research Centre for Software.

References

1. Abrahamsson, P., Babar, M.A., Kruchten, P.: Agility and architecture: can they coexist? IEEE Softw **27**(2), 16–22 (2010)
2. Adams, P.J., Capiluppi, A., Boldyreff, C.: Coordination and productivity issues in free software: the role of Brooks' Law. In: 2009 IEEE International Conference on Software Maintenance, pp. 319–328. IEEE, Piscataway (2009)
3. Adler, P.S., Kwon, S.W.: Social capital: prospects for a new concept. Acad. Manag. Rev. **27**(1), 17–40 (2002)
4. Ågerfalk, P.J., Fitzgerald, B.: Outsourcing to an unknown workforce: exploring opensourcing as a global sourcing strategy. MIS Q. **32**(2), 385–409 (2008)
5. Alvesson, M., Sandberg, J.: Generating research questions through problematization. Acad. Manag. Rev. **36**(2), 247–271 (2011)
6. Bagozzi, R.P., Phillips, L.W.: Representing and testing organizational theories: a holistic construal. In: Administrative Science Quarterly, pp. 459–489 (1982)
7. Baragry, J.: Understanding software engineering: from analogies with other disciplines to philosophical foundations. Ph.D. Thesis, School of Engineering (2000)

[22] The astute reader will notice that the map has considerable room for improvement.

[23] Question to the reader: a metaphor or analogy?

[24] Metaphor or analogy?

8. Baragry, J., Reed, K.: Why we need a different view of software architecture. In: Proceedings Working IEEE/IFIP Conference on Software Architecture, pp. 125–134. IEEE, Piscataway (2001)
9. Basili, V.R.: The experimental paradigm in software engineering. In: Rombach, H.D., Basili, V.R., Selby, R. (eds.) Proceedings of Dagstuhl-Workshop, Lecture Notes in Computer Software, pp. 1–12. Springer, Berlin (1993)
10. Basili, V.R., Zelkowitz, M.V.: The software engineering laboratory: objectives. In: Proceedings of the Fifteenth Annual SIGCPR Conference, pp. 256–269 (1977)
11. Beecham, S., OLeary, P., Richardson, I., Baker, S., Noll, J.: Who are we doing global software engineering research for? In: 2013 IEEE 8th International Conference on Global Software Engineering, pp. 41–50. IEEE, Piscataway (2013)
12. Berente, N., Recker, J.: The blank page problem. In: This IS Research Podcast (2024)
13. Bertolino, A.: Software testing research: achievements, challenges, dreams. In: Future of Software Engineering (FOSE'07), pp. 85–103. IEEE, Piscataway (2007)
14. Bird, C.: Conway's law. Making Software: What Really Works, and Why We Believe It, p. 187 (2010)
15. Boehm, B.W.: Software engineering economics. IEEE Trans. Softw. Eng. **SE-10**(1), 4–21 (1984)
16. Bonaccorsi, A., Rossi, C.: Why open source software can succeed. Res. Policy **32**(7), 1243–1258 (2003)
17. Bosch, J.: Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach. Pearson Education (2000)
18. Bosch, J.: Software architecture: the next step. In: Software Architecture: First European Workshop, EWSA 2004, St Andrews, UK, May 21–22, pp. 194–199. Springer, Berlin (2004)
19. Bossavit, L.: The Leprechauns of Software Engineering. Lulu (2015)
20. Bowman, I.T., Holt, R.C.: Software architecture recovery using Conway's law. In: CASCON, vol. 98, p. 6 (1998)
21. Briand, L.: Embracing the engineering side of software engineering. IEEE Softw. **29**(4), 96–96 (2012)
22. Brooks, F.P.: No silver bullet: Essence and accidents of software engineering. IEEE Comput. **20**(4), 10–19 (1987)
23. Capilla, R., Jansen, A., Tang, A., Avgeriou, P., Babar, M.A.: 10 years of software architecture knowledge management: practice and future. J. Syst. Softw. **116**, 191–205 (2016)
24. Card, D.N., Agresti, W.W.: Measuring software design complexity. J. Syst. Softw. **8**(3), 185–197 (1988)
25. Colfer, L.J., Baldwin, C.Y.: The mirroring hypothesis: theory, evidence, and exceptions. Ind. Corporate Change **25**(5), 709–738 (2016)
26. Conway, M.E.: How do committees invent. Datamation **14**(4), 28–31 (1968)
27. Crowston, K., Annabi, H., Howison, J., Masango, C.: Effective work practices for software engineering: free/libre open source software development. In: Proceedings of the 2004 ACM Workshop on Interdisciplinary Software Engineering Research, pp. 18–26 (2004)
28. Cruzes, D.S., Dybå, T.: Research synthesis in software engineering: a tertiary study. Inform. Softw. Technol. **53**(5), 440–455 (2011)
29. Cunningham, W.: The wycash portfolio management system. ACM SIGPLAN OOPS Messenger **4**(2), 29–30 (1992)
30. Curran, P.: Model based thinking. Quant. Useful Podcast **S2**(E05) (2020)
31. De Boer, R.C., Farenhorst, R.: In search of 'architectural knowledge'. In: Proceedings of the 3rd International Workshop on Sharing and Reusing Architectural Knowledge, pp. 71–78 (2008)
32. De Boer, R.C., Farenhorst, R., Lago, P., van Vliet, H., Clerc, V., Jansen, A.: Architectural knowledge: getting to the core. In: Software Architectures, Components, and Applications: Third International Conference on Quality of Software Architectures, QoSA 2007, Medford, MA, USA, July 11–23, pp. 197–214. Springer, Berlin (2007)

33. De Souza Santos, R., Ralph, P., Arham, A., Stol, K.J.: Distributed scrum: a case meta-analysis. ACM Comput. Surv. **56**(4), 1–37 (2024)
34. Dittrich, Y.: What does it mean to use a method? Towards a practice theory for software engineering. Inform. Softw. Technol. **70**, 220–231 (2016)
35. Dubin, R.: Theory Building, Revised edn. The Free Press (1978)
36. Endres, A., Rombach, H.D.: A Handbook of Software and Systems Engineering: Empirical Observations, Laws, and Theories. Pearson Education (2003)
37. Fagerholm, F., Münch, J.: Developer experience: concept and definition. In: International Conference on Software and System Process (ICSSP), pp. 73–77. IEEE, Piscataway (2012)
38. Foucault, M.: The Archaeology of Knowledge and the Discourse on Language. 1971. Pantheon Books (1972)
39. Garlan, D., Allen, R., Ockerbloom, J.: Architectural mismatch or why it's hard to build systems out of existing parts. In: Proceedings of the 17th International Conference on Software Engineering, pp. 179–185 (1995)
40. Garlan, D., Perry, D.E.: Introduction to the special issue on software architecture. IEEE Trans. Softw. Eng. **21**(4), 269–274 (1995)
41. Garlan, D., Shaw, M.: An introduction to software architecture. In: Advances in Software Engineering and Knowledge Engineering, pp. 1–39. World Scientific, Singapore (1993)
42. Giles, T.D.: The missing metaphor. J. Tech. Writing Commun. **31**(4), 373–390 (2001)
43. Graziotin, D., Fagerholm, F., Wang, X., Abrahamsson, P.: What happens when software developers are (un) happy. J. Syst. Softw. **140**, 32–47 (2018)
44. Graziotin, D., Lenberg, P., Feldt, R., Wagner, S.: Psychometrics in behavioral software engineering: a methodological introduction with guidelines. ACM Trans. Softw. Eng. Methodol. **31**(1), 1–36 (2021)
45. Gregor, S.: The nature of theory in information systems. MIS Q. **30**(3), 611–642 (2006)
46. Hall, J.N.: Perl: Internet duct tape. IEEE Internet Comput. **3**(4), 95–96 (1999)
47. Hassan, N.R., Lowry, P.B., Mathiassen, L.: Useful products in information systems theorizing: a discursive formation perspective. J. Assoc. Inform. Syst. **23**(2) (2022)
48. Hassan, N.R., Mathiassen, L., Lowry, P.B.: The process of information systems theorizing as a discursive practice. J. Inform. Technol. **34**(3), 198–220 (2019)
49. Herbsleb, J.D., Grinter, R.E.: Splitting the organization and integrating the code: Conway's law revisited. In: Proceedings of the 21st International Conference on Software Engineering, pp. 85–95 (1999)
50. Hirschheim, R., Newman, M.: Symbolism and information systems development: myth, metaphor and magic. Inform. Syst. Res. **2**(1), 29–62 (1991)
51. Hofmeister, C., Kruchten, P., Nord, R.L., Obbink, H., Ran, A., America, P.: Generalizing a model of software architecture design from five industrial approaches. In: 5th Working IEEE/IFIP Conference on Software Architecture, pp. 77–88. IEEE, Piscataway (2005)
52. Hsia, P., Hsu, C.t., Kung, D.C.: Brooks' law revisited: a system dynamics approach. In: Proceedings Twenty-Third Annual International Computer Software and Applications Conference, pp. 370–375. IEEE, Piscataway (1999)
53. Huber, G.P., Van De Ven, A.: Longitudinal Field Research Methods: Studying Processes of Organizational Change, vol. 1. Sage (1995)
54. Jackendoff, R.: What is a concept, that a person may grasp it? Mind Lang. **4**(1-2), 68–102 (1989)
55. Jansen, A., Bosch, J.: Software architecture as a set of architectural design decisions. In: 5th Working IEEE/IFIP Conference on Software Architecture, pp. 109–120. IEEE, Piscataway (2005)
56. Joblin, M., Apel, S.: How do successful and failed projects differ? A socio-technical analysis. ACM Trans. Softw. Eng. Methodol. **31**(4), 1–24 (2022)
57. Johnson, P., Ekstedt, M., Jacobson, I.: Where's the theory for software engineering? IEEE Softw. **29**(5), 96–96 (2012)
58. Jørgensen, M.: Myths and over-simplifications in software engineering. Lecture Notes on Software Engineering, vol. 1(1), p. 7 (2013)

59. Kendall, J.E., Kendall, K.E.: Metaphors and methodologies: living beyond the systems machine. In: MIS Quarterly, pp. 149–171 (1993)
60. Kitchenham, B.A., Dyba, T., Jorgensen, M.: Evidence-based software engineering. In: Proceedings of the 26th International Conference on Software Engineering, pp. 273–281. IEEE, Piscataway (2004)
61. Kruchten, P.: An ontology of architectural design decisions in software intensive systems. In: 2nd Groningen Workshop on Software Variability, pp. 54–61. Groningen (2004)
62. Kruchten, P.: Putting the "engineering" into "software engineering"'. In: 2004 Australian Software Engineering Conference, pp. 2–8. IEEE, Piscataway (2004)
63. Kruchten, P.: Software architecture and agile software development: a clash of two cultures? In: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering, vol. 2. ACM, New York (2010)
64. Kruchten, P., Nord, R.L., Ozkaya, I.: Technical debt: from metaphor to theory and practice. IEEE Softw. **29**(6), 18–21 (2012)
65. Lehman, M.: Programs, life cycles, and laws of software evolution. Proc. IEEE **68**(9), 1060–1076 (1980)
66. Lenberg, P., Feldt, R., Wallgren, L.G.: Behavioral software engineering: a definition and systematic literature review. J. Syst. Softw. **107**, 15–37 (2015)
67. Lerner, J., Tirole, J.: Some simple economics of open source. J. Ind. Econ. **50**(2), 197–234 (2002)
68. Leshem, S., Trafford, V.: Overlooking the conceptual framework. Innov. Educ. Teach. Int. **44**(1), 93–105 (2007)
69. MacCormack, A., Baldwin, C., Rusnak, J.: Exploring the duality between product and organizational architectures: a test of the "mirroring" hypothesis. Res. Policy **41**(8), 1309–1324 (2012)
70. Medawar, P.: Is the Scientific Paper a Fraud. PB Medawar, The Threat and the Glory. Harper Collins, New York (1990)
71. Miles, M.B., Huberman, A.M.: Qualitative data analysis: an expanded sourcebook. Sage (1994)
72. Mockus, A., Fielding, R.T., Herbsleb, J.D.: Two case studies of open source software development: apache and mozilla. ACM Trans. Softw. Eng. Methodol. **11**(3), 309–346 (2002)
73. Morgan, L., Finnegan, P.: Beyond free software: an exploration of the business value of strategic open source. J. Strategic Inform. Syst. **23**(3), 226–238 (2014)
74. Nahapiet, J., Ghoshal, S.: Social capital, intellectual capital, and the organizational advantage. Acad. Manag. Rev. **23**(2), 242–266 (1998)
75. Nisbett, R.: The geography of Thought: How Asians and Westerners Think Differently... and Why. Simon and Schuster (2004)
76. Noda, A., Storey, M.A., Forsgren, N., Greiler, M.: Devex: What actually drives productivity? Commun. ACM **66**(11), 44–49 (2023)
77. Offutt, J.: What is the purpose of publishing? Softw. Testing Verif. Reliab. **21**(4) (2011)
78. Parnas, D.L.: Stop the numbers game. Commun. ACM **50**(11), 19–21 (2007)
79. Parnas, D.L., Clements, P.C.: A rational design process: how and why to fake it. IEEE Trans. Softw. Eng. **SE-12**(2), 251–257 (1986)
80. Payne, C.: On the security of open source software. Inform. Syst. J. **12**(1), 61–78 (2002)
81. Pohl, K.: Requirements Engineering: An Overview. Citeseer (1996)
82. Qiu, H.S., Nolte, A., Brown, A., Serebrenik, A., Vasilescu, B.: Going farther together: the impact of social capital on sustained participation in open source. In: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), pp. 688–699. IEEE, Piscataway (2019)
83. Ralph, P.: Software engineering process theory: a multi-method comparison of sensemaking–coevolution–implementation theory and function–behavior–structure theory. Inform. Softw. Technol. **70**, 232–250 (2016)
84. Ralph, P.: Toward methodological guidelines for process theories and taxonomies in software engineering. IEEE Trans. Softw. Eng. **45**(7), 712–735 (2018)

85. Ralph, P., Tempero, E.: Construct validity in software engineering research and software metrics. In: Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018, pp. 13–23 (2018)
86. Redwine, S.T., Riddle, W.E.: Software technology maturation. In: International Conference on Software Engineering, pp. 189–200 (1985)
87. Rigby, P.C., German, D.M., Cowen, L., Storey, M.A.: Peer review on open-source software projects: parameters, statistical models, and theory. ACM Trans. Softw. Eng. Methodol. **23**(4), 1–33 (2014)
88. Rigdon, E.E.: Rethinking partial least squares path modeling: in praise of simple methods. Long Range Plan. **45**(5-6), 341–358 (2012)
89. Robson, C.: Real World Research. Blackwell Oxford (2002)
90. Rolland, K.H., Fitzgerald, B., Dingsøyr, T., Stol, K.J.: Acrobats and safety-nets: problematizing large-scale agile software development. ACM Trans. Softw. Eng. Methodol. **33**(2), 1–45 (2024)
91. Rombach, H.D., Verlage, M.: Directions in Software Process Research, pp. 1–63. Elsevier, Amsterdam (1995)
92. Royce, W.W.: Managing the development of large software systems: concepts and techniques. In: Proceedings of the 9th International Conference on Software Engineering, pp. 328–338 (1987)
93. Rudestam, K.E., Newton, R.R.: Surviving Your Dissertation. Sage (1992)
94. Russo, D., Stol, K.J.: PLS-SEM for software engineering research: an introduction and survey. ACM Comput. Surv. **54**(4), 1–38 (2021)
95. Ryan, I., Roedig, U., Stol, K.J.: Unhelpful assumptions in software security research. In: ACM Conference on Computer and Communications Security. ACM, New York (2023)
96. Sachs, S.: Laws of software. www.laws-of-software.com
97. Sandberg, J., Alvesson, M.: Ways of constructing research questions: gap-spotting or problematization? Organization **18**(1), 23–44 (2011)
98. Schwarz, A., Mehta, M., Johnson, N., Chin, W.W.: Understanding frameworks and reviews: a commentary to assist us in moving our field forward by analyzing our past. ACM SIGMIS Database: DATABASE Adv. Inform. Syst. **38**(3), 29–50 (2007)
99. Shapere, D.: The paradigm concept. Science **172**, 706–709 (1971)
100. Shaw, M.: Prospects for an engineering discipline of software. IEEE Softw. **7**(6), 15–24 (1990)
101. Sjøberg, D.I., Bergersen, G.R.: Construct validity in software engineering. IEEE Trans. Softw. Eng. **49**(3), 1374–1396 (2022)
102. Sjøberg, D.I.K., Dybå, T., Anda, B.C.D., Hannay, J.E.: Building Theories in Software Engineering, p. 312–336. Springer, London (2008)
103. Smolander, K.: Four metaphors of architecture in software organizations: finding out the meaning of architecture in practice. In: Proceedings International Symposium on Empirical Software Engineering, pp. 211–221. IEEE, Piscataway (2002)
104. Stamelos, I., Angelis, L., Oikonomou, A., Bleris, G.L.: Code quality analysis in open source software development. Inform. Syst. J. **12**(1), 43–60 (2002)
105. Stol, K.J., Avgeriou, P., Babar, M.A., Lucas, Y., Fitzgerald, B.: Key factors for adopting inner source. ACM Trans. Softw. Eng. Methodol. **23**(2), 1–35 (2014)
106. Stol, K.J., Caglayan, B., Fitzgerald, B.: Competition-based crowdsourcing software development: A multi-method study from a customer perspective. IEEE Trans. Softw. Eng. **45**(3), 237–260 (2019)
107. Stol, K.J., Fitzgerald, B.: Researching crowdsourcing software development: perspectives and concerns. In: Proceedings of the 1st International Workshop on CrowdSourcing in Software Engineering, pp. 7–10 (2014)
108. Stol, K.J., Fitzgerald, B.: Two's company, three's a crowd: a case study of crowdsourcing software development. In: Proceedings of the 36th International Conference on Software Engineering, pp. 187–198 (2014)

109. Stol, K.J., Fitzgerald, B.: Theory-oriented software engineering. Sci. Comput. Program. **101**, 79–98 (2015)
110. Stol, K.J., Fitzgerald, B.: The ABC of software engineering research. ACM Trans. Softw. Eng. Methodol. **27**(3), 1–51 (2018)
111. Stol, K.J., Ralph, P., Fitzgerald, B.: Grounded theory in software engineering research: a critical review and guidelines. In: 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), pp. 120–131. IEEE, Piscataway (2016)
112. Stol, K.J., Schaarschmidt, M., Goldblit, S.: Gamification in software engineering: the mediating role of developer engagement and job satisfaction. Empirical Software Engineering **27**(2), 35 (2022)
113. Stol, K.J., Schaarschmidt, M., Morgan, L.: Does adopting inner source increase job satisfaction? A social capital perspective using a mixed-methods approach. J. Strategic Inform. Syst. **33**(1), 101819 (2024)
114. Suddaby, R.: From the editors: what grounded theory is not. Acad. Manag. J. **49**(4), 633–642 (2006)
115. Suddaby, R.: Editor's comments: construct clarity in theories of management and organization (2010)
116. Tang, A., Avgeriou, P., Jansen, A., Capilla, R., Babar, M.A.: A comparative study of architecture knowledge management tools. J. Syst. Softw. **83**(3), 352–370 (2010)
117. Thomson, G.P.: The Inspiration of Science. Oxford University Press, Oxford (1961)
118. Trice, H.M., Beyer, J.M.: Studying organizational cultures through rites and ceremonials. Acad. Manag. Rev. **9**(4), 653–669 (1984)
119. Van de Ven, A.H., Poole, M.S.: Explaining development and change in organizations. Acad. Manag. Rev. **20**(3), 510–540 (1995)
120. Wasserman, A.I.: Toward a discipline of software engineering. IEEE Softw. **13**(6), 23–31 (1996)
121. Weick, K.E.: What theory is not, theorizing is. Adm. Sci. Q. **40**(3), 385–390 (1995)
122. Weyuker, E.J.: Empirical software engineering research-the good, the bad, the ugly. In: 2011 International Symposium on Empirical Software Engineering and Measurement, pp. 1–9. IEEE, Piscataway (2011)
123. Whittle, J., Hutchinson, J., Rouncefield, M.: The state of practice in model-driven engineering. IEEE Softw. **31**(3), 79–85 (2013)
124. Wieringa, R., Daneva, M.: Six strategies for generalizing software engineering theories. Sci. Comput. Program. **101**, 136–152 (2015)
125. Yourdon, E.: Managing the Structured Techniques, 4th edn. Yourdon Press (1989)
126. Zhang, Y., Stol, K.J., Liu, H., Zhou, M.: Corporate dominance in open source ecosystems: a case study of openstack. In: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 1048–1060 (2022)

Teaching Research Design in Software Engineering

Jefferson Seide Molléri and Kai Petersen

Abstract In the dynamic field of software engineering (SE), where practice is constantly evolving and adapting to new technologies, conducting research is a daunting quest. This poses a challenge for researchers: *how to stay relevant and effective in their studies?* Empirical software engineering (ESE) has emerged as a contending force aiming to critically evaluate and provide knowledge that informs practice in adopting new technologies. Empirical research requires a rigorous process of collecting and analyzing data to obtain evidence-based findings. Challenges to this process are numerous, and many researchers, novice and experienced, found difficulties due to many complexities involved in designing their research. The core of this chapter is to teach foundational skills in research design, essential for educating software engineers and researchers in ESE. It focuses on developing a well-structured research design, which includes defining a clear area of investigation, formulating relevant research questions, and choosing appropriate methodologies. While the primary focus is on research design, this chapter also covers aspects of research scoping and selecting research methods. This approach prepares students to handle the complexities of the ever-changing technological landscape in SE, making it a critical component of their educational curriculum. Finally, in this chapter, we highlight several teaching modules. These comprise active learning principles to enhance student engagement through practical activities and reflection. Following the constructive alignment framework [4], we formulate teaching modules that include Intended Learning Outcomes (ILOs) and Teaching and Learning Activities (TLAs) and suggested assessments. For ease of pedagogical application, these teaching modules are clearly highlighted at the end of each relevant subsection.

J. S. Molléri (✉)
Kristiania University College, Oslo, Norway
e-mail: jefferson.mollleri@kristiania.no

K. Petersen
Flensburg University of Applied Sciences, Flensburg, Germany

Blekinge Institute of Technology, Karlskrona, Sweden
e-mail: kai.petersen@hs-flensburg.de; kai.petersen@bth.se

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2024

D. Mendez et al. (eds.), *Handbook on Teaching Empirical Software Engineering*,
https://doi.org/10.1007/978-3-031-71769-7_4

1 Research Design at a Glance

This chapter outlines a systematic approach to designing research in SE, combining structured methodology with practical advice to help researchers effectively navigate the complexities of SE research. Additionally, we offer guidance on teaching each step of the research design process.

The approach to research design can vary significantly, generally categorized into two types, as shown in Fig. 1. *Fixed designs* are highly structured and follow a systematic progression through clearly defined phases, typical for studies that utilize structured methodologies. This offers clarity and predictability, with each phase building directly on the one before [18]. In contrast, *flexible designs* are iterative and adaptive, ideal for research projects where the scope isn't initially clear and may evolve with new findings or trends [11, 18].

The generic framework for research design depicted in Fig. 2 provides the overarching structure for this chapter. It involves nine methodological steps, covering research strategy formulation (steps 1–3), tactical decisions about the overall research method (steps 4 and 5), selection of operational methods (steps 6–8), and practical considerations (step 9).

While this framework is better suited for fixed research designs, it can also fit flexible designs with minor modifications. In flexible research design, the framework's steps are not required to be completed sequentially before data collection begins. Instead, multiple iterations may refine and adjust the research process as more data are collected and analyzed. Additionally, adjustments in the research design can also lead to changes in idea generation and other phases of the research project.

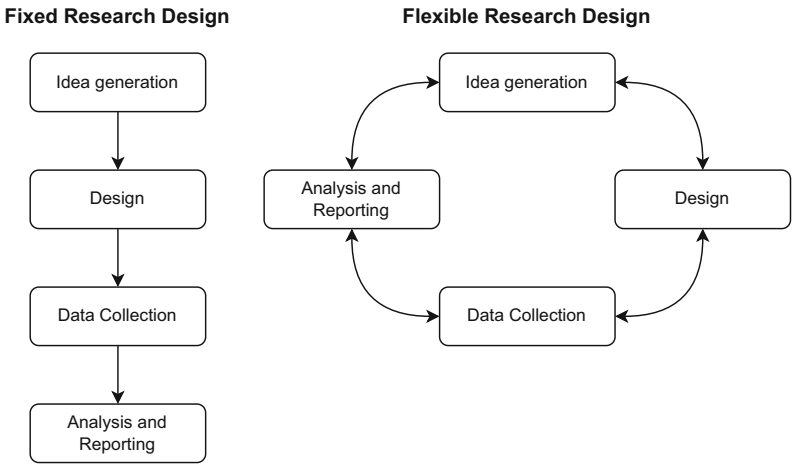


Fig. 1 Procedures for (**a**) fixed and (**b**) flexible research designs, according to [11]

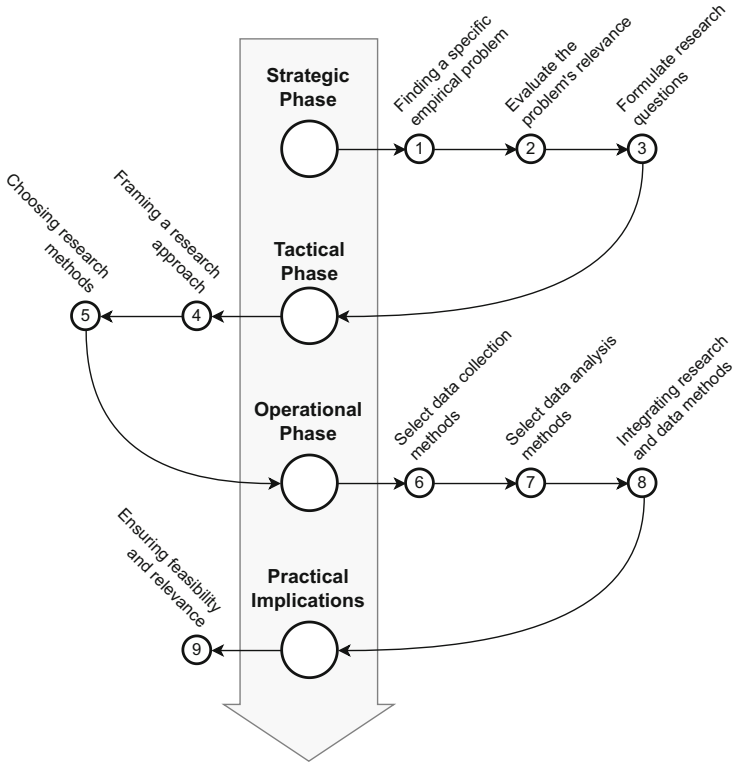


Fig. 2 Overview of the research design process

Educators should strive to teach both fixed and flexible design paradigms, providing students with the tools to choose the appropriate approach based on their research goals. Practically, researchers should consider their project's scope, the nature of the research question, and the degree of uncertainty in the data when deciding between a fixed and a flexible design.

2 Finding a Relevant Empirical Problem

Finding a relevant problem in SE requires understanding of the field's dynamic nature and how it shapes research directions. This section begins by exploring the evolving landscape of SE and its implications for research and then defines criteria for determining the relevance of a research problem.

2.1 The Current State of SE Research and Practice

Software engineering is a broad field that encompasses both academic study and practical applications. This duality, common to applied fields, offers unique advantages and challenges when identifying a potential problem that contributes to the body of knowledge. This knowledge spans the *state of practice*, i.e., practical applications and standards known, accepted, and adopted by professionals, and the *state of the art*, i.e., the recent findings and theories that have been attested by empirical research.

Integral to SE is its broad spectrum of knowledge areas, which include diverse topics ranging from software requirements and testing to software engineering management, each offering distinct perspectives and insights into the field. Many of the established knowledge areas related to SE are outlined by the Software Engineering Body of Knowledge [5].

Currently, SE research is on cutting-edge areas such as artificial intelligence (AI) and machine learning, cloud computing, DevOps, and cybersecurity. Agile methodologies, while still pivotal, are evolving to emphasize scalability and adaptability. In practice, there's a strong trend toward automating development processes and incorporating data-driven decision-making. Additionally, ethical considerations and sustainability in software development are gaining traction. This evolving landscape reflects the dynamic nature of SE and also underscores the need for research that is relevant to practice.

2.2 Sources for Identifying Problems

Essentially, a research problem is a specific issue or gap in the existing knowledge that research aims to address or resolve. Thus, a well-defined research problem typically goes hand in hand with an identified knowledge gap in the existing body of knowledge.

2.2.1 Analyzing Existing Literature

Analyzing the existing literature is a first step to identify obvious knowledge gaps. The body of knowledge represented by literature (i.e., state of the art) is a source for research problems of interest. Research articles serve as a primary source to identify problems the industry is currently facing and outline topics for further investigation. For example, a researcher interested in the challenges of integrating AI into software testing would start by examining the current literature on AI techniques within SE to determine what has been extensively studied and where gaps exist.

A thorough systematic literature review (SLR, see chapter "Teaching Literature Reviews in Software Engineering Research (pp. 531) by Sebastian Baltes and Paul

Ralph") offers a structured approach to survey the existing body of knowledge and obtain a comprehensive overview of a topic. However, researchers should consider the possible delay in academic literature reflecting the latest advancements and, thus, might also include gray literature like blogs and white papers for a more current perspective [9].

2.2.2 Bridging Theory and Practice

Researchers should extend their focus beyond academic context and actively build relationships with companies to integrate practical insights into their research design. The diverse practices and the multifaceted aspects of the software process adopted by companies are key sources for identifying current problems in SE. Therefore, researchers should seek a dialogue with industry for formulating relevant research problems.

Engaging with practitioners that have extensive experience and understanding of a particular field helps identify under-researched areas and validate the most current state of practice. For example, software development teams using agile methodologies can reveal adoption rates, challenges, benefits, and organizational impacts of these practices, offering a practical context that academic literature alone may not provide.

Suggested steps for industry engagement:

- Partner with companies through joint research and development (R&D) projects to gather data on relevant phenomena and industry needs.
- Attend industry conferences and workshops to stay updated on current practices and emerging trends and to consult with experts for practical insights.
- Share preliminary findings with industry partners and incorporate their feedback to refine research focus and ensure it addresses real-world problems.

Building on the previous example, after identifying gaps in integrating AI into software testing from the literature, researchers consulted industry experts to understand the practical challenges faced when adopting AI in testing environments. This led them to review project postmortems from development teams that had attempted AI solutions, uncovering common issues such as integration complexity and tool usability. These insights helped define a specific empirical problem: the difficulties in integrating AI for automated test generation in agile development cycles.

2.2.3 Validating Knowledge Gaps

To ensure the integrity of identified research problems, researchers should cross-reference information from various sources, like literature reviews and expert consultations. Additionally, open-access databases and industrial publications help in enhancing the trustworthiness and accuracy of the information obtained. Using

this multifaceted approach, researchers can confirm that the identified gaps are not only theoretically significant but practically relevant.

> Teaching Module 1

Intended Learning Outcome: Develop the ability to formulate a well-defined research problem in SE.

Teaching and Learning Activities: Have students develop a research proposal, including a clearly defined problem statement. This task should involve background research, stating the significance of the problem, and hypothesizing potential outcomes.

After developing research proposals, facilitate peer review sessions where students critique and provide feedback on each other's problem statements, enhancing their ability to identify and articulate research problems.

Assessment: Students will be assessed on the quality of their feedback and their ability to critically analyze the research problems posed by peers.

2.3 Criteria for Relevance

Relevance refers to the degree to which research findings are applicable and beneficial to real-world problems and practices. The concept of relevance commonly adopted in SE is inherited from information systems (IS) research [3], in which qualities, such as being of future interest of practitioners, applicable in the current state of practice, and written in a style that professionals would understand, are worthy of investigation.

Petersen et al. [14] further refine this by proposing a framework to assess industry relevance in SE research. This framework evaluates research contributions across six key aspects: what, how, where, who, when, and why, each addressing different dimensions of a project. For example, when assessing the relevance of a new testing framework for continuous integration, researchers might evaluate its application in various industry settings (where), its alignment with current testing practices (how), and its potential benefits for software quality (what). This multidimensional relevance assessment helps understand the feasibility, applicability, and generalizability of research findings in SE.

> Teaching Module 2

Intended Learning Outcome: Develop the ability to critically reflect on the relevance of research problems in SE.

Teaching and Learning Activities: Assign tasks where students must perform relevance assessment assignments and write about the relevance of a chosen

research problem (e.g., from real case studies), including the use of literature on research relevance definition.

Organize debate sessions on the relevance and impact of various research problems in SE, helping students to articulate and defend their perspectives.

Assessment: Students must submit written assignments demonstrating their ability to articulate and defend their evaluations of research problem relevance.

---

3 Formulating Research Questions

Research questions (RQs) set the direction of the investigation efforts and determine how research is conducted. They are also essential for the further steps of research design, i.e., choosing a research method and detailing a research approach. A good research question should be clear and focused, stating precisely what is intended to be studied and avoiding ambiguities. It should also address a significant gap in the existing body of knowledge, as stated in the previous step.

Consider the following poorly formulated RQ: *"How can software development be improved?"* This question is too broad and lacks specificity. It does not outline a particular aspect to be investigated, the variables, or the context involved. A more effective RQ could be the following: "How do continuous integration and deployment (CI/CD) practices impact the software release cycle time and bug detection rate in small-sized software development organizations?" This question targets a specific topic within the SE field, outlines the variables to be studied, and also defines the context by means of a population.

Well-formulated research questions not only guide the study's direction but also shed light on the most suitable methodologies to use. For example, our example of a well-formulated RQ above suggests using a quantitative approach, specific measurements like release cycle intervals and bug detection rates, and a comparison of different practices, such as the implementation versus non-implementation of CI/CD practices.

3.1 Crafting Effective Research Questions

In selecting empirical methods for SE research, Easterbrook et al. [8] distinguished different types of research questions that drive the choice of research and data collection methods. Table 1 provides an overview of the question types.

From the questions it already becomes clear that they require different methods to be answered. Descriptive-process questions focusing on detailed process

Table 1 Types of research questions, according to Easterbrook et al. [8]

Type	Structure	Example
Existence	Does [phenomenon] exist in [context]?	Does *continuous integration* exist in *small software development teams*?
Description and classification	What are the [characteristics] of [context]?	What are the *common challenges* faced by *teams* during the *adoption of agile methodologies*?
Descriptive-comparative	How does the [aspect] of [group 1] compare to [group 2]?	How does the *code quality* of *projects using test-driven development* compare to *those not using it*?
Descriptive-process	How is the [process] [action] in [context]?	How is the *deployment process* typically *executed* in *DevOps environments*?
Relationship	Is there a [relationship] between [variable 1] and [variable 2] in [context]?	Is there a *correlation* between *team size* and *productivity* in *software development projects*?
Causality	Does [event] cause/prevent the [effect] in [context]?	Does *the use of automated testing* reduce *the number of defects* in *software releases*?
Causality-comparative	Does [condition 1] cause/prevent [effect] than [condition 2] in [context]?	Does *pair programming* lead to *fewer bugs* than *solo programming* in *small-scale software development*?
Causality-comparative interaction	Does [variable 1] on [outcome] differ across [conditions]?	Does the effect of *continuous deployment* on *release frequency* vary with *team size*?
Design	What is an effective way to [action] into the [context]?	What is an effective way to *integrate security practices* into *the agile development life cycle*?

descriptions require to gather deep insights of how processes are executed and hence require detailed examinations of real processes with a variety of options (such as case studies, observations, or ethnographies). Causality questions require controlling confounding factors and hence point toward experimental studies.

Wohlin and Arum [22] introduced various decision points to arrive at a research design. They considered the definition of the research outcome (basic vs. applied), research logic (inductive and deductive), research purpose (explanatory, descriptive, exploratory, and evaluation), and research underpinning (positivist, interpretivist, constructivist, critical) as the strategy phase as input to choosing the concrete approaches for obtaining the results. These are strongly linked with the above questions.

Teaching Module 3

Intended Learning Outcome: Develop the ability to formulate and critically evaluate research questions based on given research problem formulations and goals.

Teaching and Learning Activities: Provide students with examples of software engineering problems and associated goals to perform a problem formulation

exercise. Ask students to critique and discuss the clarity and focus of these formulations, encouraging collaboration and critical thinking.

Then, have them develop a set of research questions that would effectively address the given problem and associated goal. As a role-playing activity, students take on the roles of different stakeholders (e.g., researchers and industry professionals) to formulate research questions from various perspectives.

Assessment: Students submit the set of research questions they developed, assessing clarity, focus, and alignment with the initial problem.

3.2 Frameworks for Formulating Research Questions

While the above distinction of question types helps with methodological choices, the process of formulating well-defined RQs often requires additional support from specific frameworks. These frameworks are especially useful for novice researchers, as they help in thoroughly considering all aspects of the research. Key among these are the PICO framework and the Goal-Question-Metric (GQM) approach.

PICO Framework Originally from healthcare research and often adopted in SLRs, the PICO framework aids in formulating precise research questions by focusing on four essential elements [12]:

- Population: Who are the subjects of the study?
- Intervention: What is the action or technology being investigated?
- Comparison: What is the intervention being compared to?
- Outcome: What are the expected effects or outcomes of the intervention?

The PICO framework is particularly useful for empirical studies looking at the impact of specific technologies or methodologies. For instance, to investigate the impact of agile methodologies on project success rates, a research question might be as follows: “In large tech companies (population), how does the adoption of agile methodologies (intervention) compare to the traditional waterfall model (comparison) in affecting project delivery times and productivity (outcomes)?”

GQM Approach Tailored for the SE domain by Basili et al. [2], the GQM approach is particularly useful for evaluating technologies and strategic decisions in SE. It involves a three-step process: (1) define overall goals to be achieved; (2) establish specific questions that reflect the goal; and (3) determine the metrics needed and how data will be collected to answer the question. The stepwise approach ensures that all the elements are mapped and a chain of evidence is established.

GQM is better suited for process improvement studies in SE. For example, if the goal is to assess the effectiveness of CI/CD practices, a suitable question might be as follows: “What is the reduction in deployment failures observed after

implementing CI/CD practices in software development teams?" To address this question, one would measure key performance indicators (metrics) such as the deployment frequency, success/failure rates, and mean time to recovery, both before and after CI/CD implementation.

Other Frameworks The FINER criteria evaluate the feasibility, interest, novelty, ethics, and relevance of research questions [10]. The SPIDER (Sample, Phenomenon of Interest, Design, Evaluation, and Research type) tool offers a multifaceted approach to question formulation for mixed-methods research [6].

Frameworks such as the ones listed above may not be ideally suited for interpretivist and constructivist research initially. These researches prioritize understanding the meanings, experiences, and complexities of human interactions rather than measurement or prediction. In these paradigms, RQs are broad and open-ended at the initial stages to allow for an unrestricted exploration of the topic. The 5W1H method (who, what, when, where, why, and how) supports the development of such questions and enhance their communication with a broader audience [20].

As the research progresses, these broad RQs evolve in response to new data and insights, capturing the dynamic nature of human behavior. For instance, a RQ might begin as "What processes influence software development practices in start-up environments?" and evolve into "How do agile practices impact team dynamics in tech start-ups?". Frameworks such as PICO, GQM, FINER, and SPIDER become more relevant in the later stages of research, helping to refine and structure the developed theories into more precise research questions.

4 Framing a Research Approach

Since our emphasis is on empirical research, we prioritize applied research over theoretical reasoning within this context. However, before connecting the research questions to the research logic, the purpose of the research, and the research approach, we first briefly introduce the concepts.

Research Logic *Inductive reasoning* involves making generalizations based on specific observations or experiences. This approach starts with specific data or instances and then draws broader generalizations or theories from them. *Deductive reasoning*, on the other hand, works from the general to the specific. It starts with a general statement or hypothesis and conducts studies to test the hypothesis. An example is shown in Fig. 3. *Abductive reasoning* is a third form of logical inference that seeks the simplest and most likely explanation from a set of observations. It is useful for generating hypotheses that can be subsequently tested through inductive or deductive methods. Hypotheses may also be generated from descriptive and relationship questions (induction) and later be tested with causality questions (deduction).

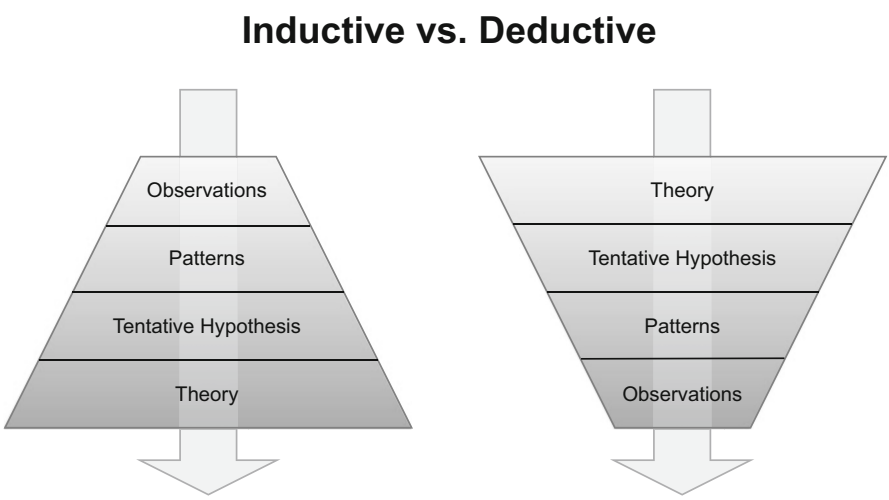


Fig. 3 Inductive versus deductive research logic

Research Purpose *Explanatory research* aims to clarify why and how there is a relationship between two aspects of a study. It focuses on understanding the causality of relationships and often follows up on findings from exploratory or descriptive research. *Descriptive research* seeks to accurately and systematically describe a population, situation, or phenomenon. *Exploratory research* is conducted for a problem that has not been clearly defined yet. That is, exploratory research aims to gain new insights and to formulate propositions and hypotheses, or more concrete research questions. *Evaluation research* is concerned with, for example, assessing the effectiveness and efficiency of a software engineering intervention.

Research Underpinnings *Positivism* assumes a single, objective reality that can be observed and measured through scientific methods. *Interpretivism*, also known as constructivism, posits that our perception of reality and to a certain extent reality itself is shaped by human and social experience and interpretation. The *critical approach* views reality as shaped by, for example, social, cultural, or economic values with a focus on critically evaluating phenomena in that reality. *Pragmatism* draws from the other underpinnings and often from abductive reasoning to emphasize that the validity or knowledge is determined by its practical utility in explaining or improving human practices.

The interconnection between research questions and the characteristics of research logic, purpose, and underpinnings is key for framing a research approach. We illustrate this interplay with examples from SE research, explaining how research logic, purpose, and underpinning align to address research questions:

RQ1: *What processes are emerging in DevOps teams to handle continuous integration and continuous deployment (CI/CD)?*

This open-ended *descriptive-process* question is suitable for an *exploratory*-purposed study that aims to generate insights via *inductive* reasoning. It aligns with an *interpretivist* underpinning as it seeks to understand emerging practices from the perspective of practitioners.

RQ2: *What are the most common types of bugs encountered in agile software development projects?*
This *description and classification* question is grounded on a *positivist* underpinning. It uses *inductive* reasoning to draw on observed data, with a *descriptive* purpose of identifying the types of bugs based on empirical evidence.

RQ3: *Does implementing systematic code reviews reduce the frequency of bugs in software releases?*
This aims to *explain* the relationship between code reviews and bug frequency, making it a *causality* question. It uses *deductive* reasoning, starting with a hypothesis to be tested, and aligns with a *positivist* underpinning for objective measurement.

RQ4: *How effective is a given new project management tool in improving team productivity compared to the previous tool?*
This *causality-comparative* question uses *deductive* reasoning to *evaluate* the impact of the tool. The *critical approach* underpinning examines the effectiveness in an organizational context.

It's important to note that research can employ a variety of combinations beyond those depicted in our examples. This inherent flexibility in research design allows researchers to effectively navigate the complexities of framing their specific research approach, facilitating the identification of suitable research question types (see Sect. 3). Vice versa, by formulating specific research questions, researchers can ensure that the research approach is consistent with the purpose, logic, and underpinning.

Teaching Module 4

Intended Learning Outcome: Develop the ability to characterize research regarding its logic, purpose, and underpinning.

Teaching and Learning Activities: Assign students articles from ESE journals for research article analysis. Have them analyze and present the research logic (inductive or deductive), purpose (exploratory, explanatory, etc.), and underpinning (positivist, interpretivist, etc.) used in these articles.

Also, in groups, students role-play as researchers planning a study. Each group is assigned a different combination of research logic, purpose, and underpinning, and they must outline a research design that aligns with these parameters.

Assessment: Students' article analyses and/or research plans are evaluated on the understanding and application of research logic, purpose, and underpinning.

5 Choosing Research Methods

Next step into the research design is choosing the appropriate methodologies and techniques that will be used for collecting, sampling, and analyzing the data. There are multiple decision aspects in this process [22], and they are highly influenced by the relevant empirical problem, knowledge gaps, and research approach established earlier.

The research process itself can be viewed as a series of interconnected steps, each employing methods and techniques with differing scopes. These range from comprehensive processes that cover all aspects of data handling to more specific techniques focused solely on one aspect of the process [19].

In this book, we have adopted a classification that aligns with the corpus of empirical studies in SE. It's important to recognize, however, that there are varied opinions among researchers regarding the classification of specific methods. In the literature, it was pointed out that methods are often misidentified by authors [16], reflecting the complexity of method selection. See, for example, the works of Stol and Fitzgerald [19] and Molléri, Petersen, and Mendes [13]; both studies employ differing approaches in classifying research methods in ESE. The former adopts a taxonomy from more mature social sciences, whereas the latter uses an inductive approach based solely on how SE authors label their own studies.

5.1 Overview of Common SE Research Methods

This section outlines common research methods in ESE, focusing on the specific contexts and scenarios where they are most effective, along with their requirements and practical considerations. It's designed to help researchers select the most suitable process for their research approach, fulfilling the methodological requirements and effectively linking them with specific research questions. Here, *requirements* refer to a short checklist that helps determine whether a study indeed meets its methodological standards; more comprehensive checklists supporting peer review can be found in Ralph et al. [17]. The research methods are organized according to the chapters of this book where they are further detailed.

5.1.1 Action Research

Action research is a method in which researchers actively engage in real-world problem-solving and practice improvement while acquiring knowledge. This approach, characterized by iterative planning, action, observation, and reflection, is particularly suitable for exploring transformations in software processes, such as adopting new technologies or modifying existing processes or redefining organizational structures. Key to action research is the active participation of both researchers and software professionals, bridging theory and practice. Best employed for testing and refining methods and tools in live settings, action research is further

detailed in chapters “Teaching Action Research (pp 391) by Miroslaw Staron and Action Research with Industrial Software Engineering: An Educational Perspective (pp 417) by Yvonne Dittrich, Johan Bolmsten, and Catherine Seidelin”.

Requirements The following questions help decide whether the planned study is indeed an action research:

- ☐ Does the study aim at solving a specific practical problem within a real-world setting?
- ☐ Does the research involve collaboration between researchers and participants who are directly affected by or interested in the problem?
- ☐ Does the study use an iterative process, involving cycles of planning, acting, observing, and reflecting?
- ☐ Does the research combine practical action with generating theoretical knowledge about the problem and its solutions?
- ☐ Is the research design flexible and adaptive to changes and new findings as the study progresses?

5.1.2 Survey

Survey is a research method for gathering data on practices, opinions, or behaviors from a broad audience, often practitioners. They are particularly effective in understanding trends, attitudes, and patterns within the community. This method is ideal to collect extensive data from a large population, making it particularly useful for understanding the prevailing practices in SE. It can also effectively compare different subgroups within the population. Surveys can be tailored to specific research questions, making them versatile for various knowledge areas. Further details about surveys are given in chapter “Teaching Survey Research in Software Engineering (pp. 503) by Marcos Kalinowski, Allysson Allex Araújo, and Daniel Mendez”.

Requirements The following questions help determine if a study is a survey:

- ☐ Is the survey designed to collect data on practices, opinions, or behaviors from a wide audience?
- ☐ Are the survey questions aligned with the research objectives and hypotheses?
- ☐ Is the survey sample representative of the target population?
- ☐ Is there a clear plan for data collection and analysis of survey results appropriate for the study design?
- ☐ Does the study ensure ethical considerations, such as consent and anonymity of respondents?

5.1.3 Literature Review

Systematic literature review (SLR) is a comprehensive overview of existing literature on a specific topic, ensuring a thorough understanding of the current state of

the art. SLRs follow a structured approach to minimizing bias in literature selection and analysis. They are crucial for identifying research gaps, summarizing existing evidence, and setting the foundation for future research. Systematic mapping studies (SMS) are broader in scope but less focused than SLRs, providing a broader overview of a research field and categorizing existing literature. Multivocal Literature Review (MLR) includes both academic and gray literature, offering a perspective from both research and practice. These methods are invaluable for consolidating knowledge in SE. More on this research method is detailed in chapters "Teaching Literature Reviews in Software Engineering Research (pp. 531) by Sebastian Baltes and Paul Ralph" and "Teaching Systematic Literature Reviews: Strategies and Best Practices (pp. 559) by Marcela Genero and Mario Piattini".

Requirements The following questions help decide whether the planned study is indeed a SLR:

- ☐ Is the review guided by a clear set of research questions or objectives?
- ☐ Does it follow a predefined and transparent methodology for searching, selecting, and appraising literature?
- ☐ Are inclusion and exclusion criteria for literature selection clearly defined and applied consistently?
- ☐ Are the credibility and relevance of literature sources critically assessed?
- ☐ Does the review synthesize findings from selected literature to answer the research questions?
- ☐ Is the review's methodology documented in a manner that allows for future replication and validation?

SMS can be assessed using a similar checklist, but the synthesis approach (see question 4) in SMS is about mapping the field's landscape rather than providing detailed answers to research questions. For MLR, we suggest adding the additional requirements:

- ☐ Does the review include a mix of academic and gray literature (such as industry reports, blogs, and white papers)?
- ☐ Is there a strategy to integrate and compare insights from both types of sources?

5.1.4 Experiment

Experimental studies are pivotal in ESE research, particularly for evaluating the applicability of methods and technologies. These studies often involve controlled settings where participants (like practitioners, organizations, or even software systems) are exposed to different treatments to observe the effects on measurable variables. Randomized controlled trials (RCTs) are the gold standard for experimental studies, but other more reliant quasi-experiment designs are also common in SE research. Field experiments are conducted in real-world settings, which adds complexity and reduces control over variables compared to traditional experiments. Further information on experiments can be found in chapter "A Course

on Experimentation in Software Engineering: Focusing on Doing (pp. 305) by Sira Vegas and Natalia Juristo".

Requirements The requirements on RCTs are relatively strict. The following questions help you evaluate whether your study is indeed a RCT.

- ☐ Does the study involve the random assignment of participants or subjects to control and experimental groups?
- ☐ Is there a control group that does not receive the experimental treatment or intervention?
- ☐ Does the study investigate the effects of a specific intervention or treatment?
- ☐ Are procedures, treatments, and measurements standardized and applied consistently across all participants?
- ☐ Are specific outcomes being measured to assess the effects of the intervention?
- ☐ Are participants, researchers, or both blinded to group assignments (single-blind or double-blind)?
- ☐ Is statistical analysis used to compare outcomes between control and experimental groups?

While quasi-experiments and field studies can be assessed using a similar checklist, they differ from RCTs in that random assignment and strict control over procedures (i.e., the first two questions) may not be feasible. In addition, blinding of participants and researchers (i.e., question 6) is less strict in quasi-experiments and not practical in real-world settings.

5.1.5 Simulation

Simulation offers a valuable approach for exploring complex systems, processes, or behaviors in a controlled, virtual environment. This method is used when real-world experimentation is impractical or impossible. Simulations allow researchers to model and analyze software development scenarios, including system performance, user behavior, or project management strategies. They provide insights through the creation and testing of virtual models, which can replicate various aspects of SE processes. Detailed information and guidelines to simulation-based studies are provided in chapter "Teaching Simulation as a Research Method in Empirical Software Engineering (pp. 367) by Breno Bernard Nicolau de França, Nauman Bin Ali, Dietmar Pfahl, Valdemar Vicente Graciano Neto".

Requirements The following questions help decide whether it is a simulation study:

- ☐ Is a realistic and representative model of a process or system developed for the simulation?
- ☐ Are the parameters and variables clearly defined and based on empirical data or credible assumptions?

- ☐ Does the simulation allow for controlled manipulation and observation of key variables?
- ☐ Are the expected outcomes of the simulation aimed at drawing conclusions about real-world practices or theories?
- ☐ Is there a process for validating the simulation results against actual scenarios or data?

5.1.6 Ethnography

In ESE, ethnography studies focus on understanding the cultural and social dynamics within live environments, such as a software project or a development team. It is a qualitative research method involving immersive observation and interaction with participants in their natural settings, gathering insights into the behaviors, practices, and interactions. It aims at uncovering implicit knowledge, workflows, and challenges that might not be evident through other research methods. Ethnography is useful when exploring the human and organizational aspects of software development, offering rich, detailed insights into the lived experiences of practitioners. Chapter "Teaching and Learning Ethnography for Software Engineering Contexts (pp. 595) by Yvonne Dittrich, Helen Sharp, and Cleidson R. B. de Souza" offers more details on ethnographic studies.

Requirements The following questions help in designing an ethnography:

- ☐ Does the study involve immersive, long-term observation within a realistic setting?
- ☐ Are qualitative data collection methods like participant observation and interviews used?
- ☐ Does the researcher engage deeply with the cultural and social context of the studied environment?
- ☐ Is there a focus on investigating human or organizational factors, such as behaviors, practices, and interactions of participants?

5.1.7 Case Study

Case study research delves into a specific research problem within their real-life context, often focusing on organizations, projects, or processes. This method allows handling complex phenomena through triangulation, which involves integrating various data sources for deeper insights. Unlike action research or design science, the researcher in a case study observes without direct involvement in the process. Designing a case study requires careful consideration of context, as highlighted by Petersen and Wohlin's [15] framework. Particularly effective in situations where the boundaries between the phenomenon and context are blurred, case studies are discussed more extensively in chapter "Teaching Case Study Research (pp 465) by Stefan Wagner".

Requirements The following questions help in determining whether a study is a case study, as discussed in Wohlin and Rainer [21]:

- ☐ Does the study describe a case? Examples for cases could be a software system, a development process, a software organization, etc.
- ☐ Does the study investigate contemporary phenomena (occurring at present time)?
- ☐ Is the study conducted in a real-life context (e.g., industry, open source), thus involving real developers, products, and processes?
- ☐ Are multiple methods for data collection used (triangulation)?
- ☐ Does the researcher act passively (only collecting data) and therefore not take an active role in the investigated case?

5.1.8 Design Science

Design science is ideal for creating innovative solutions to specific problems where existing theories are insufficient. It's about actively developing, testing, and refining new tools, methods, or models in real-world environments. Similar to action research, this method encourages a problem-solving approach and collaboration with practice, but it focuses on creating a practical and tangible solution. Its outcomes contribute significantly to advancing SE practices and methods, bridging a gap between theory and practice. Further details are provided in chapter "Teaching Design Science as a Method for Effective Research Development (pp. 481) by Oscar Pastor, Mmatshuene Anna Segooa, and Jose Ignacio Panach".

Requirements For assessing if a study is design science, use the following questions:

- ☐ Does the study aim to create a novel artifact (tool, method, model) to solve a specific problem?
- ☐ Is the artifact designed based on identified requirements and knowledge gaps?
- ☐ Does the research involve iterative cycles of development and evaluation of the artifact?
- ☐ Are the results of artifact evaluation used to refine and improve its design?
- ☐ Does the study contribute both to practical applications and theoretical understanding?

5.1.9 Grounded Theory

Grounded theory (GT) is a qualitative research method aimed at generating theories from empirical data. Primarily used in exploratory studies, it involves systematic data collection and analysis to construct theoretical frameworks that explain the observed phenomena. GT involves systematic data collection and analysis, often through techniques such as interviews, observations, and document analysis. It is

particularly useful for understanding complex human and organizational dynamics in software development, offering insights into processes, behaviors, and interactions within this field.

Requirements For assessing if a study is grounded theory, use the following questions:

- ☐ Does the study aim to develop a theory grounded in empirical data?
- ☐ Is there an iterative process of data collection and analysis?
- ☐ Are concepts and categories developed inductively from the data?
- ☐ Is there a constant comparison of data to refine and develop theoretical insights?
- ☐ Does the study emphasize theoretical sampling to explore emerging themes and concepts?

> Teaching Module 5

Intended Learning Outcome: Develop the ability to assess whether studies meet specified methodological requirements.

Teaching and Learning Activities: Provide students with a set of criteria for different research methods. Assign a selection of research papers labeled as a specific type of study, and ask them to analyze each paper to determine if it meets the methodological requirements of the given study type.

Also, students role-play as authors and peer reviewers in a research method debate. The "authors" defend their methodology choice, while "peer reviewers" critically assess the alignment with method requirements.

Assessment: Students' analyses are evaluated on the accurate assessment of methodological adherence and argumentation on methodological validity.

> Teaching Module 6

Intended Learning Outcome: Develop the ability to argue for and against the choice of research methods based on a formulated research strategy.

Teaching and Learning Activities: Students are divided into groups, each advocating for a different research method as the best fit for a given research approach (including problem formulation, study goals, and questions/hypotheses). A debate is held to discuss the merits and drawbacks of each method in relation to the strategy.

Assessment: Students are assessed based on their participation in debates and their arguments in support of or against the methods discussed.

5.2 Data Collection

This section serves as a toolkit to various data collection techniques, outlining the applications, advantages, and the types of data that each technique yields. These methods address different research challenges in ESE, offering both breadth and depth in data collection strategies.

5.2.1 Document Analysis

This method involves examining and interpreting documents to gather data. It is essential in SLRs, case studies, and sometimes in action research. In literature reviews, researchers systematically search and select academic papers and other scholarly articles to understand the existing body of knowledge on a topic. In field studies, researchers gather a variety of documents such as technical specifications, project reports, user manuals, and development logs. This method offers insights into the historical and contextual background of a research topic, contributing to a more comprehensive understanding of it.

5.2.2 Data Mining

Data mining or Mining Software Repositories (MSR) involves analyzing extensive datasets to uncover patterns, trends, and anomalies. It's particularly aligned with experimental studies and design science research. Data mining can reveal insights into development practices, code quality, and team productivity. By applying statistical and machine learning techniques, researchers can extract meaningful information from large volumes of data, offering a data-driven understanding of processes and outcomes.

5.2.3 Observations

Observations as a data collection method involve directly watching processes, practices, and behaviors as they naturally occur in real-world settings. This method is useful in case studies, action research, and ethnography, where understanding the workflow and interactions in software development environments is vital. Through observations, researchers capture nuances and dynamics that might not be revealed through direct inquiry. It allows for a deeper understanding of phenomena such as team collaboration, decision-making processes, and the actual use of tools and methods in practice, providing rich, contextual insights into the everyday realities.

5.2.4 Questionnaires

Questionnaires are a tool for systematically collecting data from a large number of respondents. It is particularly effective in survey research, where understanding broad trends, opinions, or behaviors across a wide audience is essential. Questionnaires are used to gather quantitative and qualitative data on various aspects, such as developer experiences, user satisfaction, or tool effectiveness. Their structured format allows for efficient data analysis, and they can be distributed widely, making them ideal for large-scale studies. The design of questionnaires, including question clarity and response format, is crucial for ensuring accurate and useful responses.

5.2.5 Interviews

Through direct conversation with individuals, researchers can gather detailed information about their experiences, opinions, and practices. Interviews are especially beneficial in case studies and action research, where understanding personal perspectives is crucial. It allows researchers to explore complex topics like decision-making processes, experiences with specific technologies, or perceptions of industry trends. This method facilitates a deeper, more nuanced understanding of the human elements, often uncovering information that isn't accessible through quantitative methods. With regard to interviews, the type of interview to be conducted needs to be decided:

- *Structured interviews* with a standardized set of questions that are strictly followed
- *Unstructured interviews* which are informal and do not follow a set of predetermined questions
- *Semi-structured interviews* mixing predetermined questions and those emerging during the conversation

5.2.6 Other Data Collection Methods

Beyond the commonly mentioned methods, there are other data collection techniques worth noting. **Focus groups and workshops** involve gathering a group of individuals to discuss and provide feedback on a particular topic or issue. **Delphi studies** consist of rounds of inquiry to a panel of experts, with each round using feedback to refine opinions and reach consensus on specific topics. **Artifact analysis** involves examining the software products, such as code quality, architecture, and usability. These methods provide additional lenses through which researchers can gather valuable data, complementing more traditional data collection approaches.

Teaching Module 7

Intended Learning Outcome: Develop the ability to evaluate and select appropriate data collection strategies for various SE research contexts.

Teaching and Learning Activities: Organize a data collection workshop where students must analyze different data collection methods, discussing their advantages, limitations, and applicability to various SE research contexts.

In addition, assign students to select a data collection strategy for a given research approach, requiring them to justify their choice based on the research objectives and context.

Assessment: Students' justifications are evaluated on the understanding and application of data collection methods.

5.3 Data Analysis

This section overviews various data analysis methods used in ESE research, including both qualitative and quantitative approaches. This summary helps in understanding the strengths of each approach and how they can be applied to analyze data effectively.

5.3.1 Thematic Analysis

Thematic analysis is another qualitative approach, used for identifying and interpreting patterns or themes in data. It's particularly useful in organizing and analyzing qualitative data like interview transcripts and observation notes. This method helps in understanding the underlying processes and dynamics of the phenomenon under study. ESE often employs thematic analysis to gain insights into areas like developer experiences, team interactions, or user feedback.

5.3.2 Content Analysis

Content analysis offers a more structured approach to analyzing textual data. It involves quantifying and analyzing the presence, meanings, and relationships of certain words, themes, or concepts. Researchers use content analysis to systematically evaluate communication patterns, document contents, or online discussions. This method is particularly valuable for assessing trends, attitudes, and cultural norms within the SE community, providing a quantitative perspective on qualitative data.

5.3.3 Statistical Analysis

This is a fundamental tool for interpreting quantitative data and turning it into actionable insights. The analysis involves using statistical techniques to test hypotheses, understand relationships, and draw conclusions from data. This can include descriptive statistics to summarize data, inferential statistics to make predictions or inferences from a sample to a population, correlational analysis to assess relationships between variables, and advanced techniques like regression analysis to understand the relationship between variables. This is used in experimental studies, surveys, and sometimes in case studies.

5.3.4 Machine Learning

Machine learning (ML) as a data analysis approach involves using algorithms to analyze large datasets and extract insights. It's effective in identifying patterns, predicting outcomes, and automating decision-making processes. ML can be applied to areas such as defect prediction, code analysis, and user behavior modeling. This approach transforms large, complex datasets into meaningful information, aiding in the development of more efficient and effective software engineering practices.

5.3.5 Meta-analysis

Meta-analysis is a quantitative method used to statistically combine and analyze results from multiple studies on a similar topic. It provides a comprehensive view of the existing evidence, identifying overall trends and drawing more robust conclusions than individual studies alone. Meta-analysis is particularly valuable for synthesizing findings across different research projects, thereby enhancing the understanding of the effectiveness and impact of various SE practices and tools. This method offers a systematic approach to assess the generalizability of findings and to identify areas where research is consistent or where there are discrepancies.

5.3.6 Other Data Analysis Methods

Beyond the most frequently mentioned analysis techniques, there are others worth mentioning. **Cross-case analysis** is employed in case study research to compare findings across multiple cases, enhancing the generalizability by drawing comparisons and contrasts across several scenarios. **Iterative testing and prototype analysis** involve evaluating and refining a product or process through repeated cycles, allowing for gradual improvements in proposed solutions.

Teaching Module 8

Intended Learning Outcome: Develop the ability to select appropriate data analysis methods aligned with the data type and research needs.

Teaching and Learning Activities: In a workshop, students analyze various data types (e.g., survey results, interview transcripts) to choose the most suitable analysis method and justify their selections. They also compare methods which are similar or target the same type of data (e.g., content vs. thematic analysis) and discuss their suitability in specific research contexts.

Assessment: Students are assessed on the rationale behind their choices, with emphasis on their ability to align data analysis methods with data types.

5.4 Selecting Research, Data Collection, and Analysis Methods

Having briefly presented the different research methods, for the given questions, we have a first pointer which methods may be suitable for the questions asked. Table 2 illustrates how different research question types (see Sect. 3) can be matched with potential research methods and provides a brief rationale for each method's selection. Please note that this table is not comprehensive; it serves as an example, and many other valid method-question pairings exist.

To illustrate an example where a mismatch between research question types and research methods can lead to ineffective combinations, consider using surveys for answering causality questions. While effective for gathering a broad range of data, survey has limited control of confounding factors, which makes it unsuitable for establishing causal relationships. Still, surveys can be quite useful for causality-comparative questions, where comparing perceptions or experiences across different groups can infer comparative causality.

The second step in research involves aligning research methods with suitable data collection and analysis techniques. This strategic pairing is essential for ensuring the effectiveness and validity of the research. As Wohlin and Aurum [22] suggest, research methods should be viewed as frameworks incorporating various data handling techniques. Table 3 provides typical pairings for ESE, but researchers are encouraged to explore alternative combinations tailored to their specific research needs and contexts. This adaptable approach ensures a more targeted and effective research methodology.

Moreover, the application of triangulation in research, which involves using multiple data sources, methods, or theoretical perspectives, further enhances the credibility and depth of findings. In ESE, triangulation allows researchers to validate results through different lenses, cross-verify findings, identify inconsistencies, and capture various dimensions of the research problem. For instance, when studying

Table 2 Mapping of question types to research methods with rationale

Question type	Potential research methods	Argumentation for usage
Existence	Surveys; observations	Surveys are effective for quantifying prevalence, and observations allow for firsthand witnessing of the phenomenon in its natural context
Description and classification	Document analysis; case studies	Document analysis is suitable for understanding documented properties and categorizations, while case studies provide an in-depth look at specific instances to understand their nature and classify them
Descriptive-comparative	Surveys; case studies	Surveys can gather comparative data across a large sample, and case studies allow for a detailed comparison of specific instances
Descriptive-process	Observations; case studies	Observations give a real-time view of processes as they occur, while case studies can provide a comprehensive understanding of the process within a particular context
Relationship	Surveys; interviews	Surveys can quantify and identify relationships across a broad sample, and interviews allow for in-depth exploration of perceptions and experiences regarding these relationships
Causality	Controlled experiments; data mining	Controlled experiments are ideal for testing causal relationships under controlled conditions, while data mining can uncover patterns and potential causal links in large datasets
Causality-comparative	Controlled experiments; surveys	Controlled experiments accurately compare different conditions for causal effect, while surveys can compare perceptions and experiences across different groups to infer comparative causality
Causality-comparative interaction	Controlled experiments; data mining	Controlled experiments can analyze variable interactions under specific conditions, and data mining helps in identifying complex patterns of interaction in large-scale data
Design	Design science; action research	Design science is suitable for developing and evaluating artifacts to meet desired goals, while action research allows for iterative testing and refinement in real-world settings

software development practices, combining quantitative surveys to gather broad data on team productivity with qualitative interviews to gain in-depth insights into individual experiences can offer a richer and more nuanced understanding. This approach ensures a more robust analysis that better reflects the complexities of real-world contexts.

Table 3 Typical combinations of research methods and data collection and data analysis approaches. The data techniques are ordered by their significance to the research method

Research method	Data collection	Data analysis
Action research	Observations; interviews; document analysis; focus groups	Thematic analysis; statistical analysis (descriptive, inferential)
Survey	Questionnaires; interviews	Statistical analysis (descriptive, inferential, correlational)
Literature reviews	Document analysis	Content analysis; meta-analysis
Experiment	Data mining; questionnaires; artifact analysis	Statistical analysis (inferential, correlational, regression)
Simulation	Data mining; observation; artifact analysis	Statistical analysis (descriptive, inferential)
Ethnography	Observation; interviews; artifact analysis	Thematic analysis
Case study	Interviews; document analysis; observations, questionnaires; focus groups	Thematic analysis; content analysis; cross-case analysis; statistical analysis (descriptive)
Design science	Data mining; observations; document analysis; Delphi study; artifact analysis	Iterative testing; prototype analysis
Grounded theory	Observation; interviews; document analysis	Thematic analysis; content analysis

> Teaching Module 9

Intended Learning Outcome: Develop the ability to synergize data collection and analysis methods to align with research questions, methods, and data types.

Teaching and Learning Activities: Students role-play as researchers where they must choose an overall research method, and suitable data collection and analysis techniques for a given research problem, explaining how these choices complement each other. Utilize collaborative tools like Miro or Trello to allow students to visually map out research designs.

Assessment: Students receive real-time feedback from peers and instructors on their collaborative tools. They are assessed on how well the methodological choices align with the research questions, methods, and data types.

5.5 Data Collection and Analysis in Flexible Research Designs

Flexible research designs, characterized by their iterative nature, present unique challenges and opportunities in the context of data collection and analysis. Data collection methods are not static but evolve based on insights gained during the

research process. Initial methods are chosen to address preliminary questions, but as new patterns or themes emerge from the data, researchers may expand their data collection techniques. This iterative approach includes emergent sampling, i.e., including additional participants or sources as needed, and triangulation, i.e., employing multiple data collection methods to uncover new dimensions of the research problem.

Data analysis in flexible designs is an ongoing process that begins early and evolves throughout the study. Researchers continuously reflect on data, refining research questions and adjusting methods as new insights emerge. This iterative analysis, known as progressive focusing, involves broad initial data examination that gradually narrows to focus on significant emerging themes. Effective implementation of flexible data collection and analysis allows researchers to align their analytical focus with the evolving research context and findings.

6 Ensuring Feasibility and Relevance

While the questions we wish to answer should be the main driver for the method selected, it may not always be feasible. This section explores additional factors influencing method selection, such as practical constraints and the study's relevance to current SE problems. Emphasis is given to trade-offs inherent in these choices and how they impact the decision-making process, ensuring researchers can balance theoretical aims with practical realities effectively.

Here, we would like to encourage researchers and students to be aware of such factors and make them explicit when arguing and discussing the choice of research methods, as there may be practical reasons for adjusting the choice of methods besides the research questions. They, however, should be the main driver and implications of deviations need to be well founded.

6.1 Methodological Implications

Research methods vary in their focus. Surveys, for example, provide breadth, capturing wide-ranging data from large samples, ideal for general trends. In contrast, case studies offer depth, providing detailed insights into specific scenarios or organizations. Researchers must decide between a broad overview of a topic and an in-depth understanding of a particular aspect.

Research methods also differ in how obtrusive they are to participants and their applicability in specific contexts. For instance, ethnographic studies are less obtrusive, allowing observation of natural behaviors, but are highly context-specific. Surveys, while more obtrusive, are less context-dependent and can be applied more universally [19].

Context includes environmental factors, organizational settings, and specific conditions under which the software engineering practices are carried out. Understanding the context helps in designing studies that are not only theoretically sound but also practically applicable. This involves considering specific nuances of the software development environment and tailoring research methods to suit these unique characteristics. Contextual awareness ensures that findings are reflective of, and applicable to, the real-world scenarios they intend to address.

Finally, relevance is a critical factor in ESE research. It pertains to the significance and applicability of the research to real-world problems. A relevant study addresses current challenges or gaps in SE practice, thereby contributing meaningful insights. Researchers must continually assess the relevance of their work, ensuring it aligns with current industry trends, technological advancements, and practical needs.

6.2 Practical Implications

Wohlin and Aurum [22] pointed out that factors beyond methodological ones may also influence the method choice. As an example, they present a case where a student chooses a method based on the competence of using the method. Effective research design requires acknowledging practical limitations, such as time and resources. Researchers are encouraged to realistically evaluate their capabilities against the demands of the chosen methods.

Data availability also plays a crucial role; for instance, case studies demand detailed data from specific organizations, while extensive surveys need access to large audiences. Obviously, conducting a case study requires good connections to industry. Also, we may not always be able to obtain the sample in surveys we ideally would like to have. Practically, random sampling would be desirable, but in software engineering the most common approaches for sampling are purposive and convenience sampling [1].

Choosing a method that demands extensive time and resources may not be feasible for a small team with limited funding. For instance, longitudinal studies, evaluating the long-term effects of implementing new practices, require significant investment in time and personnel. Similarly, opting for depth, such as conducting detailed ethnographic studies to understand developer interactions within teams, may limit the generalizability due to their focus on specific environments.

The trade-offs between these elements significantly impact the research design decision-making process. Flexible designs, in particular, allow for adjustments in response to changing circumstances, making the research more responsive to evolving data and insights. This flexibility is key for research topics where the scope is likely to shift, ensuring that studies reflect current trends. Researchers must carefully weigh these trade-offs to ensure their chosen methodology aligns with their research goals and practical constraints.

7 Summary

This chapter is a foundational guide for SE researchers on designing research proposals, setting the stage for further chapters in this book. It focuses on identifying relevant empirical problems, formulating targeted research questions, and choosing appropriate methodologies. It addresses challenges in method selection and classification within ESE and provides insights into various methods for data collection and analysis techniques. The chapter aims to help researchers make informed methodology choices and understand their research's practical implications. It also includes interactive and engaging learning activities for a high-education setting. The following chapters will expand on research methods and strategies and provide in-depth insights and hands-on guidance for teaching empirical research methods in SE.

7.1 Further Reading

In addition to this book's complementary chapters, readers are encouraged to explore Eastbrook et al. [8] for selecting empirical methods in SE research. Wohlin and Aurum's [22] decision-making framework, extensively referenced in this chapter, is a valuable resource we built upon. For comprehensive guidelines on various methods, Molléri, Petersen, and Mendes [13] offer an extensive compilation of methodological literature. Creswell's book [7], while more general, provides insightful guidance for research design across various fields, aiding in formulating research proposals.

References

1. Baltes, S., Ralph, P.: Sampling in software engineering research: a critical review and guidelines. Empir. Softw. Eng. **27**(4), 94 (2022)
2. Basili, V.R., Caldiera, G., Rombach, H.D.: The goal question metric approach. In: Encyclopedia of Software Engineering, pp. 528–532 (1994)
3. Benbasat, I., Zmud, R.W.: Empirical research in information systems: the practice of relevance. MIS Q. 3–16 (1999)
4. Biggs, J., Tang, C., Kennedy, G.: Teaching for Quality Learning at University 5e. McGraw-Hill Education (UK) (2022)
5. Bourque, P., Fairley, R.E.: Swebok v3. 0: guide to the software engineering body of knowledge. IEEE Comput. Soc. 1–335 (2014)
6. Cooke, A., Smith, D., Booth, A.: Beyond pico: the spider tool for qualitative evidence synthesis. Qual. Health Res. **22**(10), 1435–1443 (2012)
7. Creswell, J.W., Creswell, J.D.: Research Design: Qualitative, Quantitative, and Mixed Methods Approaches. Sage Publications (2017)

8. Easterbrook, S., Singer, J., Storey, M.-A., Damian, D.: Selecting empirical methods for software engineering research. In: Guide to Advanced Empirical Software Engineering, pp. 285–311 (2008).
9. Garousi, V., Felderer, M., Mäntylä, M.V.: Guidelines for including grey literature and conducting multivocal literature reviews in software engineering. Inform. Softw. Technol. **106**, 101–121 (2019)
10. Hulley, S.B., Cummings, S.R., Browner, W.S., Grady, D.G., Hearst, N., Newman, T.: Conceiving the research question. Design. Clin. Res. **335**, 18–26 (2001)
11. Kampenes, V.B., Anda, B., Dybå, T.: Flexibility in research designs in empirical software engineering. In: 12th International Conference on Evaluation and Assessment in Software Engineering (EASE). BCS Learning & Development (2008)
12. Kitchenham, B.A., Budgen, D., Brereton, P.: Evidence-Based Software Engineering and Systematic Reviews, vol. 4. CRC Press, Boca Raton (2015)
13. Molléri, J.S., Petersen, K., Mendes, E.: Cerse-catalog for empirical research in software engineering: a systematic mapping study. Inform. Softw. Technol. **105**, 117–149 (2019)
14. Petersen, K., Börstler, J., bin Ali, N., Engström, E.: Revisiting the construct and assessment of industrial relevance in software engineering research. In: 1st International Workshop on Methodological Issues with Empirical Studies in Software Engineering (WSESE'2024) (2024)
15. Petersen, K., Wohlin, C.: Context in industrial software engineering research. In: 2009 3rd International Symposium on Empirical Software Engineering and Measurement, pp. 401–404. IEEE, Piscataway (2009)
16. Rainer, A., Wohlin, C.: Case study identification: a trivial indicator outperforms human classifiers. Inform. Softw. Technol. **161**, 107252 (2023)
17. Ralph, P., Ali, N.b., Baltes, S., Bianculli, D., Diaz, J., Dittrich, Y., Ernst, N., Felderer, M., Feldt, R., Filieri, A., et al.: Empirical standards for software engineering research (2020). arXiv preprint arXiv:2010.03525
18. Robson, C., McCartan, K.: Real World Research. Wiley, London (2016)
19. Stol, K.-J., Fitzgerald, B.: The abc of software engineering research. ACM Trans. Softw. Eng. Methodol. **27**(3), 1–51 (2018)
20. Tattersall, A.: Who, what, where, when, why: using the 5 ws to communicate your research. In: Impact of Social Sciences Blog (2015)
21. Wohlin, C.: Case study research in software engineering—it is a case, and it is a study, but is it a case study? Inform. Softw. Technol. **133**, 106514 (2021)
22. Wohlin, C., Aurum, A.: Towards a decision-making structure for selecting a research design in empirical software engineering. Empir. Softw. Eng. **20**, 1427–1455 (2015)

Teaching Software Metrology: The Science of Measurement for Software Engineering

Paul Ralph, Miikka Kuutila, Hera Arif, and Bimpe Ayoola

Abstract While the methodological rigor of computing research has improved considerably in the past two decades, quantitative software engineering research is hampered by immature measures and inattention to theory. Measurement—the principled assignment of numbers to phenomena—is intrinsically difficult because observation is predicated upon not only theoretical concepts but also the values and perspective of the research. Despite several previous attempts to raise awareness of more sophisticated approaches to measurement and the importance of quantitatively assessing reliability and validity, measurement issues continue to be widely ignored. The reasons are unknown, but differences in typical engineering and computer science graduate training programs (e.g., compared to psychology and management) are likely involved. This chapter therefore reviews key concepts in the science of measurement and applies them to software engineering research. A series of exercises for applying important measurement concepts to the reader's research are included, and a sample dataset for the reader to try some of the statistical procedures mentioned is provided.

Key Takeaways

- Measurement means assigning numbers to phenomena in a principled and systematic way.
- Researchers choose what to measure from webs of causally-related variables.
- Minimizing random and systematic error decreases the cost, effort, and carbon footprint of research.
- *All* measurement is "theory-laden" and "value-laden," that is, dependent upon multiple concepts, theories, motivations, and perspectives.

P. Ralph (✉) M. Kuutila · H. Arif · B. Ayoola
Dalhousie University, Halifax, NS, Canada
e-mail: paulralph@dal.ca; miikka.kuutila@dal.ca; bimpe.ayoola@dal.ca; hera.arif@dal.ca

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2024

D. Mendez et al. (eds.), *Handbook on Teaching Empirical Software Engineering*,
https://doi.org/10.1007/978-3-031-71769-7_5

- Multi-item scales or multi-metric instruments and statistical measurement models are required to assess the degree to which instruments measure what they are supposed to measure (construct validity).
- Limited adoption of philosophical realism, multiple measures, and quantitative assessment of reliability and validity undermines the credibility and effectiveness of software engineering research.

1 Introduction

Learning Objectives

- Define "measurement," "measure," "metric," and "instrument."
- Give examples of "measurement," "measure," "metric," and "instrument" in a software engineering research context.
- Explain the importance of measurement.
- Give examples of common measurement problems in software engineering research.

This chapter aims to elucidate key concepts in software measurement, apply them to software engineering research, provide resources for teaching measurement theory that are not otherwise available in this context or in a concise format, and raise awareness of a broad spectrum of measurement issues.

Measurement is the assignment of numerals to objects or events according to rules [7]—often paraphrased as "the principled assignment of numbers to phenomena"—and a *measure* is the number assigned. For example, when we measure the height of a child with a measuring tape, we assign the child a height (i.e., a measure) of, say, 85 cm. This assignment is "principled" because:

- The numbers on the tape are in the same order as on a base-10 number line.
- A cm is a widely understood measure of length that comes from an international standard system of measurement (the International System of Units).
- There exists a strong cultural norm (and de facto standard in health research), regarding how children's heights are measured: child standing straight, heels flat on the floor, barefoot, measuring from the floor to the top of the head, etc.

In other words, much care, rigor, and effort have gone into creating and validating systems that allow us to assign meaningful heights to children.

For our purposes, a *metric* is a method, algorithm, or procedure for assigning one or more numbers to a phenomenon. For example, a metric for non-comment lines of code in a Java file might involve counting semicolons. Meanwhile, an *instrument* is a (physical, conceptual, or virtual) tool for measuring something

such as a tape measure (physical), a code quality metrics API (virtual), a heart rate monitor (physical and virtual), or a personality questionnaire (conceptual). Question-based instruments are sometimes called *scales*. The extent to which an instrument measures what it purports to measure is called *measurement validity*. For instance, a test intended to evaluate intelligence may actually measure rote learning or memorization skills [32]. In a software engineering context, the science of measurement is sometimes called "software metrology" [15].

Software engineering research—indeed, computing and engineering research more generally—is plagued by measurement problems including:

- Failing to consider that metrics, instruments, scales, etc. may not measure the target property, or may conflate it with other properties (e.g., GitHub Stars conflate quality with popularity).
- Confusing metrics with properties (e.g., assuming that the number of issues in an issue tracker *is* the number of issues with the product).
- Failing to evaluate the preciseness and consistency of a metric's results; ignoring measurement error.
- Conflating perceived measures with observed measures (e.g., a manager's reported perceptions and developer's productivity may differ substantially from direct observations).
- Assuming measures are "objective."
- Operationalizing multifaceted variables with individual metrics (mono-method bias).
- Selecting problematic metrics, for example, CPU time is a poor proxy for energy consumption when a program's energy consumption strongly depends on (a) whether it uses multi-threading, (b) whether it runs on the CPU or GPU, and (c) whether it runs on performance cores or efficiency cores.
- Failing to assess, conceptually *and* quantitatively, the validity of a study's measurement strategy.
- Assuming that measures are valid because they are popular.

Measurement is important because measuring the wrong things, or measuring the right things the wrong way, creates research that superficially seems trustworthy but isn't. Poor measurement makes research more expensive and time-consuming, leads to incorrect conclusions, and undermines the credibility of not only individual researchers but also entire fields of scholarship.

> Exercises

Choose a research topic.

This chapter includes a series of exercises (in boxes like this one) to help the reader apply the concepts to their thesis or research program.

All subsequent exercises assume the reader has a specific research topic in mind, so the first exercise is to choose a topic.

If the reader has no idea what research topic to use, they can adopt or adapt one of these examples:

- The impact of immediate feedback for Android game developers in Australia on their code's energy consumption.
- The causes and consequences of burnout among female software engineers in Sri Lanka.
- The effect of AI code generation on the quality of unit tests in open-source Java projects.

Undergrads should invent topics they personally find interesting, while graduate students should stick with the (perhaps preliminary) topic of their research proposal.

Topics should (1) focus on causal relationships, (2) identify specific groups of subjects or objects, (3) identify one or more specific properties of those subjects or objects, and (4) be statements, not questions.

Our suggested exercises are intended for informal use in a flipped classroom, not formal assignments for grading. Exercises can be completed solo but usually it's better for students to work in pairs or groups. Students should write down their answers (because writing helps us be more specific) and share with a group or whole class (because explaining things aloud improves sensemaking.)

This chapter therefore explores some of the key concepts in software metrology. Understanding these concepts will help researchers envision better measurement strategies, leading to more trustworthy research. Along the way, we provide helpful exercises for learners and tips for instructors. This chapter is accompanied by an online supplement including sample data for students to practice many of the techniques mentioned (see Supplementary Materials on page 138).

Exercises

Transform your topic into a research question.

Here, *research question* denotes a broad concern driving a program of research. Many students will generate poor examples of research questions like "Does the frequency of Scrum meetings affect perceived intrateam communication?" This is just a hypothesis ("Scrum meeting frequency is directly proportional to perceived intrateam communication") rephrased as a question. Research questions should be simultaneously broader in scope and more specific regarding where and when the phenomenon of interest takes place and possibly who is involved. A better example of a research question might be "How do practices associated with Distributed Scrum affect success in software projects conducted anywhere in the world since the advent of Distributed Scrum in the mid-2000s" [52].

Students should share their research questions and discuss how to improve them.

1.1 Lessons Learned

The first author has taught research methods for management, computing, and engineering, at multiple levels from undergrad to junior faculty, including short courses, long courses, and tutorials, on a range of topics from a general introduction to research to advanced courses in quantitative methods to courses specifically on measurement. While every offering is unique, one major challenge teaching measurement stands out: getting students to *care* about measurement. Teaching measurement means asking students to fight the system—to eschew the corner-cutting status quo for more rigorous methodology. Construct validity seems abstract and unimportant compared to getting statistical significance or outperforming the latest benchmark.

To connect with students, instructors need to explain how improving measurement and construct validity helps address students' core concerns, rather than simply creating new problems. Throughout this chapter, we therefore elevate examples of how better measurement can address students' concerns.

2 Selecting Variables from the Causal Web

! Learning Objectives

- Explain what is meant by a causal "chain" or "web."
- Draw an example of a causal web.
- Analyze the implications of selecting dependent variables from different parts of a causal web.

Suppose we have built a testing tool that identifies bugs. We believe that identifying bugs is a worthwhile outcome not because it is intrinsically valuable but because of its expected effects downstream. Helping developers find more bugs presumably helps them fix the bugs, which leads to higher software quality, happier users, increased sales, and so on. We imagine a causal chain from our dependent variable to more obviously important, higher-level outcomes.

However, most links in our inferred causal chain probably have multiple antecedents (causes) and consequences (effects). User satisfaction, for instance, probably depends on a system's user experience design, feature selection, and revenue model as well as bugginess, and affects market share, intention to continue using, the development company's odds of being bought, and the probability that the software will continue to be maintained, among other things. So it is more like a causal web.

When we design a quantitative software engineering study, we typically choose dependent variables from a causal web leading up to overall software engineering

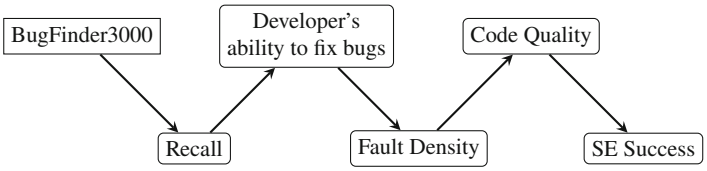


Fig. 1 Simple example of causal chain

success [48]. Where we aim in the causal web has profound implications for the difficulty and epistemic value of our research. Choosing variables too far away in the web can make some methodological approaches (especially lab-based experiments, quantitative simulations, and benchmarking studies) intractable. But choosing a variable too close in the causal web limits the epistemic value (how much we learn) and practical importance of our work.

Exercises

Look at your research question. Make a list of variables you might want to measure. Include dependent or endogenous variables, covariates, and exogenous variables (but not independent variables).

Here, we assume the instructor:

1. has introduced all the variable types mentioned above; and
2. will help students determine whether they are manipulating independent variables or measuring exogenous variables.

Back to our bug finding tool example, suppose we imagine the causal chain shown in Fig. 1. We hypothesize that our tool will have better recall than existing tools; that is, it will find more bugs. This will increase software developers' ability to fix the bugs. As developers fix more bugs, the fault density of their systems will decrease, leading to higher code quality, which contributes to overall software engineering success.

Exercises

Look at your list of variables and try to draw a causal chain or web connecting your variables to overall software engineering success (or a similar top-level construct).

Students may need a little help identifying all the relevant mediating variables.

Selecting "recall" as our dependent variable makes our study more tractable. We can get an open-source code corpus and test whether our tool can find some bugs that other state-of-the-art tools miss. However, this kind of simulation is essentially a nonexperimental design with a single participant: the researcher. We won't know

if other software developers will adopt or can even use our tool. Our tool might find bugs that are too hard to fix, have little appreciable effect on code quality, or not practically matter for the success of the project. Focusing on recall therefore limits our study's epistemic value.

In contrast, selecting overall SE success as our dependent variable decreases feasibility. We simply cannot recruit a representative sample of hundreds of software teams, randomly assign them to a treatment group and a control group, ask the treatment group to use our tool, and measure differences in overall success for, the next 5 years. Therefore, even if it were possible, selecting overall SE success as our dependent variable would make our research too slow and expensive unless we adopt a qualitative approach like action research (See chapter "Teaching Action Research by Miroslaw Staron").

The most compelling studies involve dependent variables that are far enough up the web to be ambitious, but not so far as to be impractical. Engaging some human participants to try our bug finding tool, even if those participants are undergraduate students, would have greater epistemic value than just trying it ourselves.

When a research area is dominated by de facto standard dependent variables (e.g., recall), it discourages students from imaging higher-level alternatives. In such cases, ask yourself (or your students), *why* do we care about this variable? We might care about recall because better bug finding tools have better recall. This raises the question: what are the other dimensions of quality for this kind of tool?

> Exercises

If you're having trouble moving beyond a single dependent variable, ask yourself why you care about this variable? What broader construct is this variable a dimension of? Draw a diagram showing the other dimensions of this construct.

Students may need some example diagrams (e.g., Fig. 2).

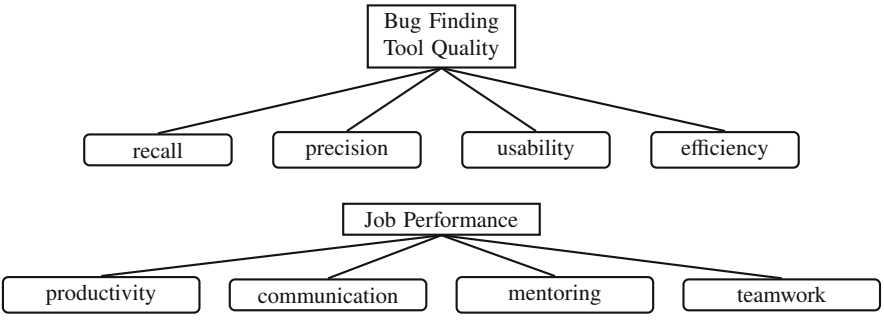


Fig. 2 Examples of abstracting to a broader construct of interest

This issue of fixating on a single dimension of a multidimensional construct of interest is not limited to tool testing. Human factors research in software engineering also tends to fixate on individual dimensions of broader concerns (e.g., productivity is just one dimension of job performance).

Moving from a single dependent variable to a comprehensive conception of success typically demands more complex, ambitious studies. However, these studies are more likely to produce interesting results that are easier to publish. A new artifact doesn't have to outperform existing artifacts on every possible dimension. Maybe the BugFinder3000 doesn't have better recall, but presents its results in a way that is more helpful to developers, so they're more successful in practice. Maybe your new way of running retrospective meetings doesn't make anyone more productive but improves communication and teamwork. The more quality dimensions you consider, the more likely you'll find something exciting.

Note: we are not advocating p-hacking. Don't go overboard and select 100 dependent variables. But if the construct you care about has five dimensions, try to measure all five dimensions.

To summarize, we choose our dependent (or endogenous) variables from a complex, often poorly understood causal chain (or web) that, we imagine, runs from our independent (or exogenous) variables to high-level concerns like overall software engineering success, human prosperity, or social cohesion. Choosing a dependent variable too close to our independent variable limits epistemic value; choosing a variable too far away makes quantitative, and especially lab-based, research intractable. We need a middle ground. When someone struggles to move beyond too-close dependent variables, ask why we care about this variable? The explanation usually reveals a broader construct of interest, of which the identified variable is just one dimension.

Once we have identified the things we want to measure, we can begin working through how to measure them in a reliable and valid manner.

3 Assessing and Improving Reliability

! Learning Objectives

- Describe the Representational Theory of Measurement.
- Define reliability, random error, systematic error, and measurement invariance.
- Explain how random error increases the cost of research.
- Explain how systematic error undermines measurement validity.
- List techniques for improving reliability.

The Representational Theory of Measurement (RTM) posits that "a measurement scale [or metric] is a many-to-one mapping—a homomorphism—from an empirical

to a numerical relational structure, and measurement is the construction of scales [or metrics]" [58]. In other words, measurement involves constructing a numerical representation of something. RTM aims to offer a principled and structured approach to convert subjective assessments into objective measurements [21, 39].

Exercises

Look at the list of variables you want to measure. How can you measure these variables? What instruments, scales, tools, methods, or metrics can you use?

RTM gives us many concepts for assessing and improving *reliability*. "Reliability is the consistency of measurement, or stability of measurement over a variety of conditions in which basically the same results should be obtained" [12]. Another way to think about reliability is as a measure of a method's resilience to different kinds of *measurement error*.

3.1 Minimizing Measurement Error

Suppose we want to measure the time it takes for an athlete to complete a 100m sprint. The coach uses a stopwatch, starting the timer when the athlete begins the sprint and stopping it when the athlete crosses the finish line. This approach to measurement may suffer from both random and systematic error.

Random error refers to the variability in measurement results that is caused by unpredictable and uncontrollable factors [54]. In our sprint example, physical button-clicking is not perfect. The coach might be off by $1/10^{\text{th}}$ of a second or so.

Exercises

Look at your list of variables. Can you identify any potential sources of random error in your chosen measurements?

If students are adamant that their measures cannot possibly have random errors, see Sect. 3.5.

Systematic error refers to a consistent, repeatable error associated with faulty equipment or flawed study design. In our sprinting example, suppose the coach always stands halfway between the starting line and the finish line. Because of the viewing angle, the coach tends to click start a little late and stop a little early. This will create a systematic error where observed times are, on average, a little less than actual run times.

Exercises

What are possible sources of systematic error in your chosen measurements?

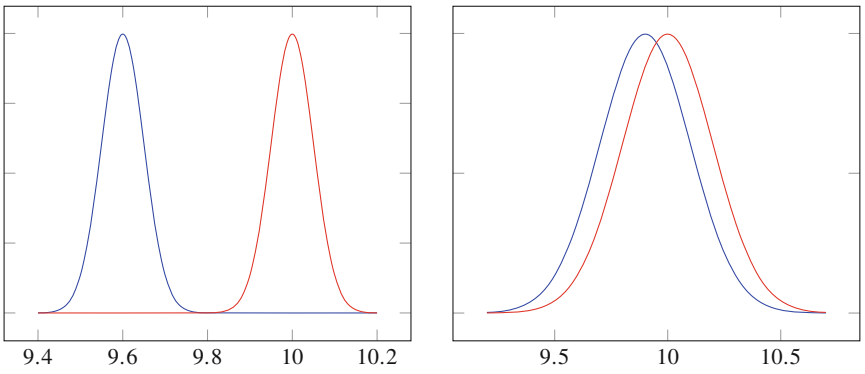
If students are adamant that their measures cannot possibly have systematic errors, see Sect. 3.5.

One way to think about random and systematic error is called *True Score Theory*. True Score Theory posits that an observed quantity (X) consists of the true quantity (T) plus or minus some amount of random error (e_r) and some amount of systematic error (e_s):

$$X = T + e_r + e_s \tag{1}$$

The difference between the true value and the observed value is the measurement error.

Random error makes your research more expensive and reduces your chances of detecting real effects. Suppose we have invented super running shoes and have recruited some sprinters to test them against regular running shoes. Further suppose that we expect our new running shoes to shave 0.4 seconds off a 100m sprint time and our stopwatch-wielding judge has a standard random error of 0.05 seconds. Figure 3a shows the probability distribution of our judge's likely observations. That is, if the true score is 9.6 seconds, our judge will probably record an observation between 9.5 and 9.7 seconds, and if the true score is 10 seconds, our judge will probably record an observation between 9.9 and 10.1 seconds. These intervals don't overlap, so we should be able to detect our expected effect.



(a) Good reliability (b) Poor reliability

Fig. 3 Sufficient (**a**) vs. insufficient (**b**) reliability given expected effect size of super shoes (blue) over regular shoes (red)

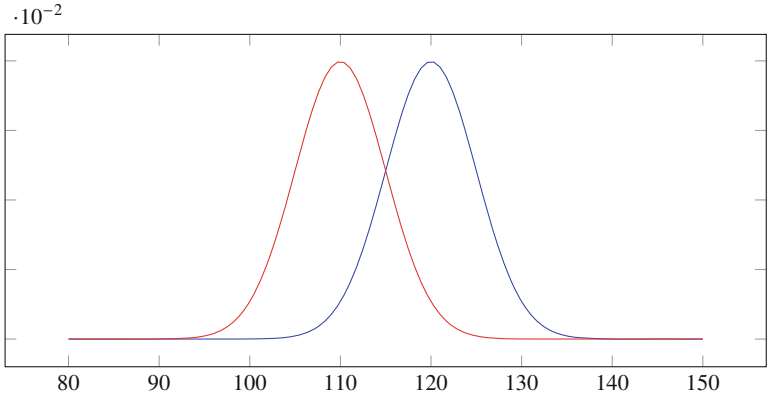


Fig. 4 Systematic error in heart rate monitor. True heart rate (blue) vs. measured heart rate (red), underestimating by 10 bpm

But what if our new shoes are only 0.1 seconds faster and our human-with-a-stopwatch has a higher standard error—say 0.2 seconds. Figure 3b shows the probability distribution of our judge's likely observations. See all the overlap? Now, due entirely to random error, we might *observe* that our super shoes are *slower* than regular shoes, even though the opposite is true.

In situations where random error is large compared to expected effect size, we either need to run heaps of trials and average the results, or employ numerous judges and average their scores, or both. With enough observations, the random errors cancel each other out, and we get more accurate estimates of the true score (the well-known "law of large numbers"). However, consider the less known (because we just made it up) "law of bad measurement is expensive": more judges + more participants = more time + more money. While this example is relatively simple, the same principle applies to more sophisticated statistical approaches.

Systematic error, unlike random error, biases results in one direction leading to consistent and repeated inaccuracies. Suppose, instead of a sprinter, we have a marathon runner. Our runner uses a special watch to measure their heart rate, and follows a sophisticated running program where they have to maintain a certain heart rate for a certain duration. Now suppose that the heart rate monitor consistently underestimates true heart rate by 10 beats per minute (bpm). This could sabotage their whole training program. The runner will constantly push too hard, and won't understand why they can't maintain the target heart rate for the expected duration.

Figure 4 illustrates the probability distributions of the true and measured heart rates. The heart rate monitor persistently underestimates the true heart rate by 10 beats per minute. As a result, we might *conclude* that the runner's heart rate is consistently lower than it actually is, even though this isn't the case.

Systematic error can cause numerous problems, including:

- Measurements that are consistently too high or too low compared to the true value.
- Drawing incorrect conclusions from seemingly consistent patterns.
- Using more complicated (i.e., harder to understand and explain; easier to mix up; sometimes less sensitive) statistics. Addressing systematic error often requires more advanced statistical techniques such as mixed effects models or structural equation modeling, which are harder to interpret, more prone to being misapplied, and sometimes less sensitive.

3.2 Measurement Invariance

Continuing with our marathon runner heart rate example, suppose that the reliability of our heart rate sensor is inversely correlated with heart rate: the higher the heart rate, the less accurate the readings [37]. In other words, measurement *varies* across the range of the phenomenon being measured.

What we want is measurement *in*variance. That is, we want our instruments to measure the same thing, in the same way, across experimental groups or the range of the phenomenon of interest. If we're comparing performance of our running shoes at different running speeds, but our performance measurement (e.g., heart rate) works differently at different speeds, it can distort our results. Similarly, if we use a questionnaire to investigate gender (or cultural, or age-related, etc.) differences in the perceived comfort of our shoes, and the questions have different meanings to people of different genders (cultures, ages, etc.), it can mess up our results.

A classic example of this is the measurement of childhood depression. In a longitudinal study following children into early adulthood, we might have a series of agree-disagree items like "the child cries easily." A 19-year-old male who cries easily is more likely suffering from depression than a 6-year-old female who cries easily.

> Exercises

Could any of your chosen measurements exhibit measurement variance? How?

A comprehensive review of statistical approaches for assessing measurement variance is beyond the scope of this chapter. Here, we simply want to encourage researchers to reflect on their instruments potential for measurement invariance. Measurement invariance is especially important in longitudinal research (where the meaning or accuracy of a measurement changes over time), questionnaire-based research (where questions have different meanings for different people), and research involving biometrics (where many instruments are only reliable

under specific conditions). For more information about establishing measurement invariance, see Putnick and Bornstein [45].

3.3 Types of Reliability

There are three main types of reliability: inter-rater, test-retest, and internal consistency.

3.3.1 Inter-rater Reliability and Agreement

Inter-rater (a.k.a. inter-judge, inter-reviewer) reliability is the extent to which different (typically human) observers give *consistent* estimates or ratings [17]. *Inter-rater agreement* is the extent to which different (typically human) observers give *identical* estimates or ratings.

In our sprinting examples above, we wouldn't expect multiple coaches with stopwatches to give identical estimates in milliseconds. Therefore, we might assess their "inter-coach" reliability using Cronbach's alpha, a statistic used to evaluate the reliability of a test based on the average correlation among items within the test [59]. We would expect good reliability (e.g., $\alpha > 0.8$), not merely statistical significance.

While alpha is very popular, it makes several problematic assumptions [1] that newer procedures, including McDonald's omega and composite reliability, overcome [9, 65]. Furthermore, if our raters were classifying data into nominal categories (e.g., brand of running shoes) or ordinal categories (e.g., self-reported running ability: beginner, intermediate, advanced), we would expect identical ratings, and a measure of inter-rater agreement would be more appropriate.

We assess inter-rater agreement with adjusted statistics, such as Krippendorff's Alpha [33], and unadjusted statistics such as percent agreement. Krippendorff's Alpha is generally preferred; however, there exist situations such as imbalanced data where it will produce nonsense and percent agreement is superior.

In software engineering, we frequently see inter-rater agreement in systematic reviews, where multiple raters must apply selection criteria.

3.3.2 Test-Retest Reliability

Test-retest reliability refers to the stability of a test from one measurement session to another using the same sample. Software engineering professionals often encounter test-retest reliability issues. Suppose we run an automated test, like a unit test, on some code, and the test passes. Then, without making any changes to the code, we run the test again, and it fails. Programmers call these unstable tests "flaky"; scientists call them "unreliable." Poor test-retest reliability often results from random error.

3.3.3 Internal Consistency

Internal consistency arises when we use multiple methods to assess the same property. It is the extent to which all the items in a multi-item scale measure the same behavior or characteristic [59] and is often assessed using Cronbach's alpha. Having several coaches with stopwatches does not constitute "multiple methods," but one method applied by multiple people. Using multiple methods is more like measuring sprint time using a person with a stopwatch, a transponder attached to the runner, and lasers at the start and finish. This might be a bit silly in the context of sprinting but using multiple methods alleviates systematic error. Appendix 1 provides an example of assessing internal consistency of class size metrics; the script and dataset are available (see Supplementary Materials).

3.4 Improving Reliability

Techniques for improving reliability include the following.

- **Pilot test instruments.** Administering instruments on a small sample helps to evaluate the feasibility and appropriateness of research instruments and methods, and to identify and fix practical or technical issues, before conducting a major study.
- **Train observers thoroughly.** In studies involving multiple research assistants making observations, developing a standard protocol and rigorously training assistants helps improve uniformity in data collection and measurement.
- **Use scripts rather than entering, cleaning, or transforming data manually.** Manually entering or transforming data is error-prone; automated scripts improve accuracy, traceability, auditability, and our ability to fix mistakes quickly.
- **Double-entering data.** Data that must be entered or labeled manually should be independently entered or labeled by at least two researchers. The resulting datasets can then be compared to resolve discrepancies. Reconciliation should be documented carefully and automated where possible. Manual labeling should be performed iteratively, calculating reliability after each round. As disagreements are resolved, researchers should clarify their labeling rules to avoid similar disagreements in subsequent rounds.
- **Statistical procedures for modeling error (mixed and random effects models).** Fixed, random, or mixed effects models can be used to account for systematic errors, random errors, or both, respectively.
- **Multiple measures and triangulation.** Using multiple measures can enhance reliability by allowing for data triangulation (see Sect. 5).

Exercises

How can you improve the reliability of your chosen measurements?

3.5 Lessons Learned

Computer science and engineering students tend to grasp easily the ideas of reliability and True Score Theory and how unreliable measures increase research cost and effort. However, they often struggle with the possibility that *their* (computerized, mostly deterministic) instruments are unreliable. The near-perfect test-retest reliability of *some* deterministic software running on the same inputs in carefully controlled laboratory settings creates the illusion of reliability. Discussing diverse examples can help. For instance:

- At the time of writing, research involving large language models (LLMs) is in vogue. As long as LLM output is nondeterministic, any measurement process using them is unreliable.
- One student gave a great example of measurement invariance when measuring soil conditions using sensors that connect to satellites. The cloudier the sky, the more likely data is missing, leading to systemic bias in temperature and moisture readings.
- Reviewing causes of flaky tests (see [42]) is a great opportunity to discuss a measurement issue that's highly relevant to industry and emphasizes the fact that deterministic computerized measurement can still be unreliable.

In contrast, some researchers have very simple instruments, for instance, measuring the quality of open-source projects based on the number of stars they have on GitHub. This example, which has great reliability but terrible validity, makes an excellent segue to our next topic.

4 Understanding Validity from a Realist Perspective

Learning Objectives

- Explain in broad strokes how realism differs from positivism and interpretivism.
- Define and give examples of *theory-laden* and *value-laden*.
- Define *generative mechanism*.
- Explain how to decide whether to *treat* a variable as latent.
- Give examples of strategies for coping with the theory-laden and value-ladenness of observation.

The previous section focused on reliability—the degree to which measures are consistent. But what if you reliably measure the wrong thing? To understand *validity*—the degree to which an instrument measures what it is supposed to measure—we need to discuss a little philosophy. Unfortunately, most computer scientists get little exposure to philosophy of science, and when they do, they tend to learn about outdated epistemological approaches like positivism and falsificationism. However, the whole idea of validity is rooted in philosophical *realism*, not *positivism*.

4.1 The Problem of Induction

Usually, the whole point of writing a scientific article is to make some claim about the world like "The new BugFinder3000 finds more bugs than the old BugFinder2000," "Distributed Scrum has no impact on project success" [52], or "the four dimensions of software sustainability are environmental, social, economic and technical" [38]. The fundamental question in the philosophy of knowledge (epistemology) is how we can justify such claims about the world. The key problem with such justification was formulated by Hume:

> *Thus, not only our reason fails us in the discovery of the ultimate connexion of causes and effects, but even after experience has inform'd us of their constant conjunction, 'tis impossible for us to satisfy ourselves by our reason, why we shou'd extend that experience beyond those particular instances, which have fallen under our observation. [26]*

In other words, suppose that every time we observe a software team transition from Waterfall to Agile, their project succeeds. We want to claim that the Waterfall-Agile transition *causes* success. But how do we know that the pattern we see in the transitions we observed also holds in transitions we did not observe (e.g., elsewhere, under different conditions, or in the future)? This is called the *Problem of Induction* because Hume questions the inductive leap from 'we observe this pattern in instances A, B, and C' to 'this pattern exists in unobserved instances X, Y, and Z.' Put another way, Hume questions how we figure out whether nature is uniform across space and time. The various epistemological schools that emerged in the twentieth century can be differentiated according to their approach to Hume's challenge.

4.2 From Positivism to Interpretivism

Positivists (or, more properly, "logical empiricists") such as Rudolph Carnap argued that we justify scientific claims by finding supporting empirical observations. The more observations we have confirming the pattern, the greater the probability that the pattern will hold in the future. In hindsight, this doesn't address Hume's chal-

lenge at all. Over time, the positivists realized that no amount of past observations conforming to a pattern could prove the pattern would continue to hold, and many of them abandoned positivism as unworkable.

Falsificationists, led by Karl Popper, accept that induction is not justified but deny that science is inductive. They view scientists as searching for observations that refute (or "falsify") causal claims. This doesn't address Hume's challenge either. For a single observation to refute a theory, "it must be presupposed that the course of nature will not change so that the experimental and observational context in which the refuting observation statement is true ceases to be true" [2, p. 91]. In other words, falsificationism assumes that nature is uniform—the very thing Hume questioned.

Failing to address Hume's Problem of Induction isn't the only issue with positivism and falsificationism. Back then, philosophers thought about cause and effect as "constant conjunction" (as in Hume's quote above); they thought x causes y if and only if, whenever x occurs, y follows. Most scientists now think of causality probabilistically: x causes y if $p(y|x) > p(y|!x)$. Positivists and falsificationists also thought that individual theories could be isolated for testing. But, as argued by Imre Lakatos, science is characterized by constellations of interconnected theories such that no one theory can be tested (and confirmed or falsified) in isolation. Furthermore, when an observation appears to confirm (or refute) a theory, it may be because the study was flawed, the observation was recorded incorrectly, the instruments were unreliable, the math connecting the observation to the theory was done wrong, or the expected observation was derived incorrectly from the theory. Lots of things can go wrong, and the same thing can go wrong over and over, so no number of confirmatory or disconfirmatory observations can verify or falsify a theory. (This is called the Quine-Duhem thesis.)

Crucially, positivists and falsificationists assumed that observation was unproblematic and measures were inherently valid. Construct validity is not, and never has been, a positivist quality criterion; the whole idea of construct validity is rooted in realism (below). Positivism is as dead as a philosophical movement can be [43]. Falsification is, similarly, an epistemological cul-de-sac because its proponents have utterly failed to address the conceptual challenges presented to it.

Many software engineering researchers are instead attracted to pragmatism—the philosophical movement associated with Charles Sanders Peirce, William James, and John Dewey. Pragmatists basically argue that truth is inextricable from usefulness—we should judge theories (and measurement instruments) based on how they enable social progress. Contemporary pragmatists might argue that induction is justified *because it is useful*, regardless of the uniformity of nature, and the realist Bhaskar makes an analogous argument (see below). But if nature isn't uniform, a theory that appears useless in one place and time might be useful in another, and vice versa. Regardless, pragmatism is unhelpful for software metrology because pragmatists would argue that measurement reliability and validity are secondary to usefulness, *thus presupposing reliable and valid measures of usefulness* without advancing substantive approaches to reliability or validity.

Moving on, interpretivists (and postmodernists) respond to Hume by agreeing that induction isn't justifiable; therefore, scientists should focus on understanding

the meaning people ascribe to their experiences instead of searching for universal laws. Interpretivists tend to prefer qualitative research methods, generate detailed accounts of specific events without generalizing to other events (past or future), and fashion their interpretations of events into internally coherent concepts and theories. Interpretivism makes sense but is deeply unsatisfying. If we want to know how a specific group of software professionals *feel* about something (e.g., generative AI, burnout, sustainability, mutation testing), interpretivism is great. But it doesn't help us compare multiple competing technologies (or practices or theories). From an interpretivist perspective, your account and my account of the same events can be completely different and yet equally valid as long as they are both internally coherent, regardless of whether either account corresponds to real events, people, or objects.

4.3 Realism

This brings us to *scientific realism*—the view that unobservable structures and processes postulated by science exist in the real world, whether or not humans exist to imagine them—and *critical realism*, Roy Bhaskar's sweeping philosophical project that aims to address Hume's Problem of Induction by charting a course between positivism and interpretivism. Realism entails many new ideas that are essential to contemporary approaches to measurement validity.

First, Hume, Carnap, Popper, and their contemporaries thought causality exists *in our minds*. Realism, in contrast, posits that causality exists in the real world, independent of human observers. Objects have powers and can influence each other (e.g., the sun has the power to warm the Earth, software has the power to frustrate users).

Second, echoing the inverted thinking of pragmatism mentioned above, Bhaskar argues that nature must be *somewhat* uniform because science is so successful. If physical and social reality was totally unstable, we wouldn't be able to successfully land a dune buggy on Mars, treat skin cancer, reign in inflation, or improve kids' reading skills. Obviously, science and engineering are not *always* successful and some things change, but nature is *somewhat* uniform, especially in the short term.

Realists argue that our assertions of causal relationships are justified by two things: the magnitude and quality of the body of empirical evidence supporting the relationship and the degree to which we understand the *generative mechanism*. The generative mechanism is *how x* causes *y*. Suppose we want to claim that the degree of formality in the presentation of software desiderata (x) reduces design creativity (y). We conduct an experiment that shows that x and y are inversely related, shows that x precedes y, and controls for third variable explanations [41]. Realists argue that, to justify our claim, we still must explain *how* desiderata presentation affects creativity. We need a different kind of study: in this case, one that shows that the more formal desiderata presentation discourages critical thinking [40]. Because we understand *how x* causes *y*, we can be more confident the relationship will endure.

Third, realists believe that many of these generative mechanisms (e.g., critical thinking) cannot be observed directly. Indeed, realists believe that reality is full of ("latent") structures that produce observable *effects*, but cannot be observed directly. These structures exist in both natural science (e.g., quasars, dark energy, the Earth's mantle) and social science (e.g., culture, personality, socioeconomic class).

4.4 Latent Variables

Recall that positivists assumed observation was unproblematic. Want to know how many ducks are in the pond? Count them. Sure you might miss a couple or count the same one twice, but it's not that complicated. Compare counting ducks to assessing the morale of a software team. You can't see morale. You can make up some survey questions that *aim* to measure morale, but how do you know they don't accidentally measure something else, like well-being, or conflate morale with other factors?

Quantifiable properties of latent processes and structures, like morale, are called *latent variables* or *constructs*. Since we cannot observe constructs directly, we must estimate them from variables we *can* observe (see the chapter on theorizing in software engineering research).

> Exercises

Look at your list of variables. Which are latent? Which are directly measurable? Are you sure?

Students may need help disentangling latent variables from common proxy measures. If you can't figure it out, say it's a good example of a tough call, and come back to it in the next exercise.

Scientists have created many methodological and statistical approaches for investigating the validity of instruments for measuring latent variables (see Sect. 5). For now, the key point is that differentiating between less problematic observations (regular variables) and more problematic observations (latent variables) is important because problematic observations demand construct validity assessment (see Sect. 5).

Statistically, variables are either latent (problematic) or not (unproblematic), but some realists (particularly Sayer [53]) argue it's more of a spectrum. *All* measurement is *theory-laden*; that is, predicated upon theoretical assumptions made by researchers when designing their studies and choosing what and how to measure [34, 57]. For example, to count the bugs found by the BugFinder3000, we need a theory of what is and is not a bug. Herzig et al. [23] defined a *bug* as a "request for corrective code maintenance," but the IEEE defines a bug as a(n):

1. "manifestation of an error in software",
2. "incorrect step, process, or data definition in a computer program"

3. "situation that can cause errors to occur in an object" or
4. "defect in a system or a representation of a system that if executed/activated could potentially result in an error". [27, p. 179–180]

None of the IEEE standard definitions have anything to do with requesting corrective code maintenance. Researchers with different theories of bugs may disagree about how many bugs the BugFinder3000 finds. Even individual *concepts* come with their own assumptions or expectations. Our measurement of bugs is predicated on myriad theoretical concepts including software, source code, software behavior, errors, users, programmers, and expectations. If two people have different concepts of "user," for example, they might disagree on the nature of bugs.

Exercises

Look at your list of things to measure. What theories and concepts underlie your research? If no theory is obvious, ask yourself, if this were an exam question and you ***had*** *to identify a related theory, what theory would you choose?*

Similarly, all measurement is *value-laden* [53]; that is, predicated upon the motivations and justifications of researchers [64]—basically what we think is important. For example, users and developers often disagree about what is and is not a bug. When our statistical analysis package gives an error message that seems clear to the developer but confusing to the user, the former might claim it's desired behavior, while the latter claims it's a usability bug. Much software engineering research ignores these value conflicts. For example, when Herzig et al. [23] suggested guidelines for classifying issue reports as bugs vs. non-bugs, they totally sidestepped the issue of who, exactly, agrees whether the reported behavior is desirable or not. Just because someone dislikes a behavior enough to write a bug report doesn't mean all relevant stakeholders will agree that the behavior should be changed.

Exercises

Look at your list of things to measure. What values underlie your research? Whose perspective are you adopting? Who might disagree with your conceptualizations?

People tend to underestimate how different other reasonable people's perspectives can be. This exercise is best done in pairs of students with different research areas, where one plays devil's advocate for the other.

So it's not so much that all variables are either directly observable or latent. It's more like all observation is predicated upon certain theories, concepts, and values. The less confident we are in the theory or the less people agree on the values involved, the more problematic the variable, and the more important it is to treat the variable as latent.

For example, when a study participant gives their age as 28 years, we're pretty confident in the underlying theories and values (the Gregorian calendar, meaning of a year, etc.). We have widespread agreement within the scientific community on the meaning of statements like "the mean age of participants was 31 years." Therefore, we treat age as a direct measurement and assume it is *inherently valid*.

In contrast, the performance of our BugFinder3000 is predicated on many theories in which we should not be nearly so confident and many values upon which people disagree. Since theory development in software engineering research is lacking [28, 56], most of our measures are problematic. The trouble is that when you are immersed in the norms and perspectives of a specific research area, it's easy to overestimate consensus. Many articles simply count bugs as if everyone agrees on what a bug is. Discussing your ideas with people from very different academic backgrounds (e.g., sociologists, historians, psychologists, microbiologists) can help surface your implicit assumptions. Adopting an explicit theory to guide your work also helps.

Because all observation is theory- and value-laden, theory is essential for research. Without theory, construct validity is meaningless, and researchers implicitly assert (with no evidence) that everyone agrees with their perspective and all of their measures are intrinsically valid. For more on validity threats from the critical realist perspective, see Johnston and Smith [29].

4.5 Summary

To summarize, all observation is problematic, but some measurements are more problematic than others. Realism challenges us to address the theory- and value-laden nature of measurement by treating more problematic variables as latent. This gives rise to construct validity—the degree to which instruments measure what we intend them to measure. Researchers doing predominately quantitative studies cannot sidestep construct validity concerns by simply claiming to be positivists, falsificationists, or pragmatists.[1] It is indefensible to simply assume that all our measures are unproblematic without any coherent response to the thesis that all observation is theory- and value-laden.

4.6 Lessons Learned

Teaching realism is quite challenging. Critical realism is a huge philosophical project with a whole mess of novel concepts and jargon. Its seminal works presuppose extensive knowledge of philosophy of science and Bhaskar, in particular,

[1] Qualitative researchers *can* dismiss construct validity concerns by embracing interpretivism, but claiming to be an interpretivist means you're not doing predominately quantitative research.

is as impenetrable as he is profound. Furthermore, some of the critical realism's key implications are inconvenient for researchers and cast doubt on large swaths of computer science and engineering research. Sayer's argument that *all* measurement is theory- and value-laden cannot be sidestepped by claiming that a study is positivist rather than realist. But students may not want to rock the field's dominant positivist boat, let alone sink it.

To cope, we recommend (1) sticking to the most important and relatable concepts, as described above, while perhaps sidestepping some of the more esoteric points, and (2) focusing on modest steps toward realism, for example, adjusting our methods better to understand and explain *how* hypothesized causal relationships manifest (i.e., generative mechanisms). Most reviewers will not be offended when a paper seeks to understand why a relationship is observed. Similarly, one can consider and specify the concepts underlying their measurement strategy without directly claiming that everyone else's measurement is irreparably broken.

Many students struggle to identify the concepts and theories underlying their work. This is often due to a lack of immersion in relevant psychological and sociological theory. There is no easy remedy for this. At the graduate level, students and graduate programs just need to prioritize reading.

5 Model-Based Theory of Measurement

! Learning Objectives

- Describe model-based approaches to measurement and statistical measurement models.
- Explain why measurement models are needed to assess technical properties of software systems.
- Explain the difference between formative and reflective measurement models, with examples.
- Describe the general strategy for evaluating a set of reflective indicators.

5.1 Statistical Measurement Models

Embracing the reality that observation is theory- and value-laden means that:

1. for every property we wish to measure, we must decide whether to treat it as unproblematic or latent; and
2. for every property we treat as latent, we must *investigate* the validity of our instruments, scales, or metrics.

not consider. not discuss. *Investigate*.

To investigate construct validity, we need a *measurement model* in which each latent property is operationalized as the shared variance of multiple indicators. As Tal explains, "According to model-based accounts, measurement consists of two levels: (i) a concrete process involving interactions between an object of interest, an instrument, and the environment; and (ii) a theoretical and/or statistical model of that process, where "model" denotes an abstract and local representation constructed from simplifying assumptions. The central goal of measurement according to this view is to assign values to one or more parameters of interest in the model in a manner that satisfies certain epistemic desiderata, in particular coherence and consistency" [58].

In other words, a model-based approach to measurement acknowledges the existence of latent structures and processes and proposes a way to measure them by theorizing relationships between latent variables (more problematic) and indicators thereof (less problematic). We need multiple indicators for each latent variable because the degree to which the indicators of X converge with each other and diverge from indicators of other latent variables provides evidence that the indicators do, in fact, measure X.

5.2 Technical Properties of Software Systems Are All Latent

In software engineering research, we usually see statistical measurement models used with questionnaire surveys. Job satisfaction, for example, might be measured using a questionnaire comprising MacDonald and Maclntyre's 10-item generic job satisfaction scale [36]. We rarely see statistical measurement models in repository mining, benchmarking, or other lab-based quantitative research. (Using machine learning to predict a variable based on some training data is not a measurement model.)

However, most technical properties of software systems are just as problematic (theory- and value-laden) as psychosocial phenomena like job satisfaction.

Consider the size of a software system. Size metrics are predicated on numerous theoretical concepts such as source code, code "lines," functions, methods, classes, packages, libraries, function points, etc. Size metrics are also predicated upon value judgments. For instance, when measuring the size of software system, should we include the programming language's standard library or third-party libraries? What about services or microservices upon which the system depends? Size is value-laden because people with heterogeneous perspectives, contexts, and goals might reasonably answer these questions differently.

Behavioral properties of software systems (e.g., efficiency, responsiveness) and common code quality dimensions (e.g., understandability, maintainability) are similarly problematic. So are code smells (code characteristics often indicative of underlying problems). For instance, the *Long Method* code smell is predicated on theoretical concepts including source code, method, code line, etc. Furthermore, we can unambiguously define any method exceeding 80 non-comment lines of code

as "long," but the threshold selection and exclusion of comments are value-laden choices. Even if everyone agreed, we remain faced with value judgments about whether each long method actually is or indicates a problem.

No one is advancing a substantive argument for treating properties of software systems as unproblematic. While previous articles have advocated for more sophisticated measurement approaches (e.g., [16, 49, 50]), many individual studies continue to ignore construct validity.

Acknowledging the theory- and value-laden nature of observations of not only psychosocial phenomena but also technical properties of software systems clarifies the need for model-based approaches to measurement. Now, we can begin taking construct validity seriously using measurement models.

5.3 Types of Measurement Models

Approaches to measurement modeling include common factors [5], partial least squares (PLS) regression, and forged concepts [35]. All three of these approaches can be used with multidimensional variables, support structural equation modeling, and involve dimension reduction. They all use two-tier models where higher-order (latent or emergent) variables are inferred from lower-order (less problematic) variables. They differ in their estimation of and assumptions about the nature of higher-order variables.

Choosing among these approaches is difficult because leading experts disagree about the relative merits of each approach and the circumstances under which each approach is appropriate. Our aim here is to advocate for measurement models in general and highlight some of the key issues involved. We will focus on the common factor model not because it is best, but because we understand it better than the others.

5.4 The Common Factor Model

The common factor model, like realism, posits that unobservable structures (like personality) *cause* observable effects (like specific human behaviors) in our world.

Imagine we are studying the job satisfaction of software testers. Suppose for simplicity that each tester uses just one of many available tools for identifying bugs. We hypothesize that the quality of the bug finding tool affects their job satisfaction (Fig. 5). The single arrow indicates the hypothesized causal relationship.

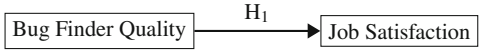


Fig. 5 Hypothesized latent structure

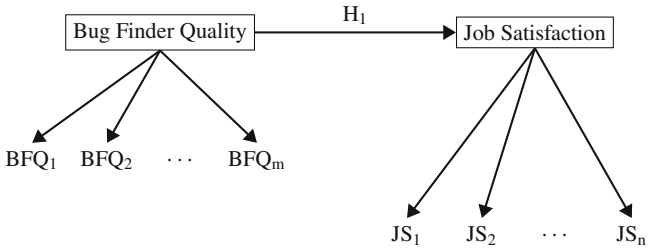


Fig. 6 Hypothesized latent structure with reflective indicators

The technical phenomenon of tool quality, the psychological phenomenon of job satisfaction, and the causal structure linking them are *all* latent; that is, their measurement is intrinsically problematic. To measure our two latent variables, then, we need to find some properties that are less problematic (i.e., less theory- and value-laden) to observe. In the common factor model, these less problematic variables are called *reflective indicators* because they reflect the latent variable of interest. We can then formulate a measurement model [22] such as the one shown in Fig. 6.

There's a lot going on in this measurement model. You can see our hypothesis, H_1, that Bug Finder Quality causes Job Satisfaction. Bug Finder Quality (BFQ) is operationalized using m reflective indicators labeled $BFQ_1 \ldots BFQ_m$. Job Satisfaction is operationalized using n reflective indicators labeled $JS_1 \ldots JS_n$. (Naming the indicators like this helps keep your datasets organized later.)

See how the arrows point *from* the latent variable *to* the reflective indicators? That's because the common factor model assumes that the latent variable that we can't see *causes* changes in the indicators that we can see. The changes in the indicators *reflect* changes in the underlying construct [44].

We estimate the latent variable in terms of the *shared variance*[2] of its reflective indicators because the latent variable is the factor that all the indicators have in common—hence the name "common factor model."

The usefulness of our measurement model therefore depends on: (1) the quality of our reflective indicators, and (2) the way we estimate our latent variables from them.

Exercises

Select two or more constructs from the list of things you'd like to measure. Identify at least three reflective indicators for each. Draw a measurement model showing your constructs, indicators, and hypotheses.

Students may need help identifying potential reflective indicators. Discussing their models in groups may help.

[2] A mathematically simple way to get the shared variance is to sum the indicators, but more sophisticated approaches, such as confirmatory factor analysis, are typically used.

5.5 Good Reflective Indicators

The best kind of reflective indicators are those comprising comprehensive instruments that have been used widely and validated repeatedly in lots of different contexts similar to the context at hand. Making and validating instruments is quite difficult, so using existing ones saves time and effort. Fortunately, good instruments exist for many psychological and psychosocial constructs. For example, it's not hard to find a questionnaire instrument (a.k.a. a *scale*) for job satisfaction that has been used in dozens of studies and undergone both quantitative and qualitative evaluations of its psychometric properties. Unfortunately, good instruments do not exist for many technical constructs like the quality of a bug finding tool.

In a good scale, the reflective indicators:

- measure different things that are all driven by the construct;
- are highly but not perfectly correlated with each other (convergent validity);
- are not too correlated with indicators of other scales in the measurement model (discriminant validity); and
- Cover all aspects or dimensions of the phenomenon of interest (content validity).

Good questionnaire scales have additional properties. For example, including both direct (e.g., "My work is interesting") and reversed (e.g., "I feel bad about my job") items improves construct validity and mitigates response bias. While many software metrics are also reversed (e.g., LCOM metrics measure *lack of* cohesion), it probably doesn't matter because the ostensible benefits of including reversed items are all about the psychological effect they have on a person filling out a questionnaire. Regardless, inverting the values of reversed indicators at the beginning of your analysis helps avoid confusion.

Technical metrics can be so highly correlated that they cause multicollinearity problems and some statistical analyses will fail. Multicollinearity can be addressed by dropping one or more indicators. If two indicators are nearly perfectly correlated, you don't lose any meaningful information by dropping one of them.

Beyond that, the meanings of "highly correlated" and "not too correlated" are relative to context. One approach is to generate a correlation matrix for all of your indicators. Then, divide them into two groups: correlations between items in the same scale (A) and correlations among items in different scales (B). The *smallest* member of A should be *larger* than the *largest* member of B. When doing this comparison, either take the absolute value of all the correlations or invert any reversed indicators before you start.

However, you can't just use *n* minor variations on the same question or measure. For example, you would not have a questionnaire with items like "I am happy with my job"; "I am pleased with my job"; "I like my job"; "My job is great"; etc. These semantically identical items will create the illusion of validity. You need semantically diverse indicators, the answers to which will all be driven by the same construct. For example, some of the items in MacDonald and Maclntyre [36] scale

are "I feel secure about my job," "My wages are good," and "I get along with my supervisors." These questions are semantically different.

Similarly, to measure the size of a software system, we don't count lines of code, non-comment lines of code, logical lines of code, lines of code including third-party libraries, lines of code excluding third-party libraries, and so on. These metrics are too similar. We need to count *different* things that are all driven by the size of the system like number of methods, number of fields, number of classes, one lines-of-code (LOC) variant, etc.

These items should cover all aspects of the latent variable. With unidimensional variables, this just means asking lots of semantically diverse questions or selecting diverse technical metrics. Multidimensional variables are a little trickier.

5.6 Multidimensional Variables

When we operationalize a construct using several reflective indicators, it's called a *reflective measurement model*. Modeling something like software quality reflectively is problematic because it has several dimensions that may be weakly correlated (e.g., usability and carbon footprint) or inversely correlated (e.g., precision and recall, effectiveness and efficiency).

Suppose we have three latent variables, Accuracy, Efficiency, and Usability (of the Bug Finder tool), and a *second-order* latent variable, Bug Finder Quality, as shown in Fig. 7.

Further suppose we have separate, multi-item scales (or multi-metric instruments) for each of Accuracy, Efficiency, and Usability. These may include subjective measures (e.g., of perceived efficiency), objective measures (e.g., benchmark results), or both.

If the reflective indicators for Accuracy, Efficiency, and Usability are all highly intercorrelated, we don't need to model them as separate dimensions [13]. We just combine all three subscales and go back to the model shown in Fig. 6. However,

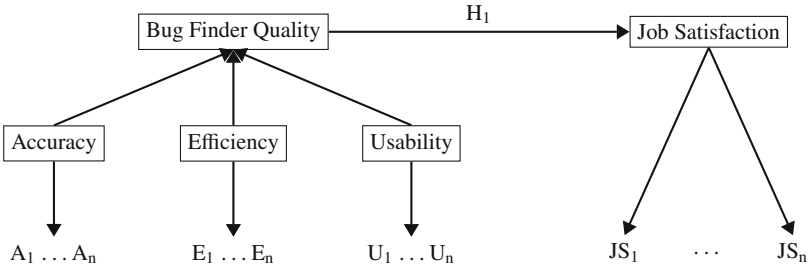


Fig. 7 Hypothesized latent structure with multidimensional variable

if Accuracy, Efficiency, and Usability aren't highly correlated, or are inversely correlated, we need something else.

And here is where it gets messy. This situation is difficult to model in covariance-based SEM, which is probably why formative models are underused (i.e., formative constructs are often modeled as reflective) [13]. Some people use "Mode B" in PLS to model formative relationships, but what PLS is actually doing, mathematically, may or may not be a sensible way to model the formative relationship in question. A host of other approaches to modeling composite and emergent variables have been proposed (e.g., forged concepts [35]), but there is no consensus about when to use what.

Here's what we can say with confidence. A causal-formative measurement model posits that the dimensions (e.g., efficiency) *cause* the higher-order construct (e.g., Bug Finder Quality) [10]. Thus, the ideas of convergent and discriminant validity are less helpful in formative models. Whereas reflective measurement models consider measurement error at the indicator level, formative measurement models consider measurement errors at the construct level. When the only multidimensional variable in an experiment is the dependent variable, and it has a small number of dimensions (e.g., $d \leq 5$), it may be preferable to run d separate structural equation models—one for each dimension.

Further mathematical details of the different approaches to estimating second-order latent variables are beyond the scope of both this chapter and introductory research methods courses. Here, we just want to emphasize that (1) including multidimensional constructs significantly increases analytical complexity, and (2) researchers should not invent their own half-baked math for estimating composite constructs.

> Exercises

Look over your list of things you'd like to measure. Are any of them multidimensional? What are their dimensions?

5.7 Lessons Learned

Most software engineering students are comfortable enough with mathematics and algorithms to quickly grasp the idea of operationalizing a latent variable as the shared variance among several reflective indicators. They quickly grok that it's like a weighted average. However, many students struggle to apply this idea to their work, especially when it's not the norm in their subfield. For example, many software engineering researchers build tools to identify bugs automatically and evaluate their effectiveness using a measure of recall. They don't even bother with precision because they reason that the user can manually disregard false positives. Recall isn't latent and the case for multiple measures is murky.

We can address this by focusing on *their concerns* rather than a principled argument about rigor. Researchers worry that their studies won't produce any significant results and will be harder to publish. If the researcher backs up the conceptual hierarchy and chooses a more multifaceted (and latent) view of success, it creates more opportunities for significant results.[3] Perhaps their bug fixing tool will underperform existing tools on recall but is more efficient, has higher precision, and is easier to incorporate into commercial build systems. Showing that the new system outperformed previous systems *on some dimension* increases the chances of publishing the paper.

Publishing papers based on whether statistical significance was achieved is, of course, totally unscientific. Regardless, more comprehensive success measures are good for science. For example, making software more environmentally sustainable is much more difficult if sustainability is not considered as part of success.

Despite usually being good with math, students universally struggle with the idea of evaluating convergent and discriminant validity by comparing the smallest intra-scale correlation to the largest inter-scale correlation. Instructors will need to demonstrate, and students should try this themselves in an assignment or lab setting.

6 Instrumentation and Scaling

Learning Objectives

1. Explain how to create a multi-metric scale to measure a property of a software system.
2. Use factor analysis to assess and improve the convergent and discriminant validity of a unidimensional, multi-item scale or multi-item instrument.

6.1 Instrumentation for Psychometric Properties

One of the most popular (and often misunderstood) methods of creating a questionnaire instrument to measure a psychological attribute was proposed by Likert [6]. A *Likert scale* is **NOT** asking a respondent to agree or disagree with a statement on a five-point scale. Instead, a Likert scale is a multi-item summative scale designed using Likert's rigorous scaling method.

[3] Even when correcting α for multiple comparisons, the more success dimensions we evaluate, the more likely we will find one on which new tool excels.

Many detailed accounts of Likert's scaling method are available (e.g., [62]), so we won't repeat the process here. Briefly, though, Likert scaling involves clearly defining the construct of interest, generating 80–100 possible items, and multiple rounds of pilot testing and statistical analysis to select the best subset of items. Further rounds of qualitative analysis to assess the psychometric properties of the scale are recommended. This process can require 30 or more human participants. Once the resulting scale has been used in studies and shown to have predictive validity (i.e., predict or be predicted by what we theorized), we are even more confident in our operationalization[4] of the construct.

Alternative psychometric scaling approaches including Thurstone, Guttman, and semantic differential are equally complicated. Simply making up some questions is not acceptable.

6.2 Instrumentation for Software Properties

This section describes one good way of developing an instrument to measure a property of a software system. This is not the only way; rather, it exemplifies the level of rigor that should pervade instrumentation. Our recommended approach looks laborious because it is. However, when researchers apply such rigorous instrumentation approaches, they typically get to write an instrument development paper (which becomes part of a graduate student's thesis) and have more confidence in their subsequent findings.

6.2.1 Define the Constructs of Interest

First we need a clear definition of the construct we want to measure. Common words with multiple meanings like "complexity" or "understandability" are not sufficient. For example, defining maintainability as "the ease with which a software system can be modified, repaired, or enhanced over time" would be problematic because it sets up maintainability as a mutual property of a system and a person (the maintainer). You need to think deeply about what *exactly* you're trying to measure. Ask yourself not only "what am I trying to measure?" but also "from whose perspective?" and "in what context?" What are the different kinds or dimensions of the property you want to measure? What is its range? Try to define it using common everyday language. If you must use jargon, the jargon must also be clearly defined in common, everyday language or technical terms that have standard or widely agreed meanings.

[4] "Operationalize" is a common term in the literature on construct validity. It refers to how we measure the construct, including our instruments and statistical approach.

> Exercises

Choose one of the constructs you'd like to measure and define it as precisely as possible. Discuss your definition in small groups. Look for ambiguity in each others' definitions.

6.2.2 Select Related Constructs

Assessing construct validity works better when our construct is situated within a small group of similar constructs. This helps us ensure that our construct differs significantly from related constructs. For example, when measuring the complexity of a software system, we'd want to ensure that complexity wasn't conflated with system size, coupling, or cohesion. If one of our coupling metrics is correlated more with some cohesion metrics than with our other coupling metrics, we'd get suspicious. Therefore, we should select a few (no more than five) related constructs. If we can think of more than five, select the most closely related.

It's ok if we believe these constructs are correlated or causally related. Correlation between constructs doesn't mean that they measure the same thing.

It's not ok if two constructs are essentially the same thing. For example, we would not include "trust in organization" and "perceived risk of dealing with organization" because these are arguably semantically equivalent.

> Exercises

Brainstorm a small set of related constructs.

6.2.3 Select, Vary, or Create Metrics

Software engineering researchers have proposed many metrics that purport to measure important properties of software systems. For example, productivity metrics include lines of code changed, function points implemented, number of issue tickets resolved, number of commits made, and number of files changed. Using a well-validated, multi-metric instrument for one of the related constructs will save time and help your work build on the cumulative body of knowledge.

For each construct that doesn't have well-validated, multi-metric instrument available, generate a long list of possible metrics. The exact number depends on how complex the construct is, but aim for *at least* 20. We can generate metrics in at least three ways:

1. Find metrics that have already been proposed for this construct.
2. Create new metrics that seem like plausible indicators of the construct.
3. Create variations.

For example, suppose we want to measure the size of an object-oriented codebase. First, we look at metrics calculated by existing tools or proposed by prior research: non-comment lines of code, numbers of classes, methods, fields, etc. If there are many such metrics and they seem to cover the entire range of the construct, perhaps that's sufficient. However, brainstorming novel metrics is a useful exercise even if it seems like we have enough. For size, we might imagine things like number of unit tests, lines of documentation, or number of user stories implemented. We continue creating new, diverse metrics until we have good content validity; that is, we've covered all areas or dimensions of the construct.

Each metric we find or create should be scrutinized for face validity—does this make any sense? Using app store star ratings as a code quality metric doesn't make sense "on its face," because most of the people leaving those ratings never see the code.

Once we are confident in our face and content validity, we can generate variations. For example, when computing non-comment lines of code in Java, we could have variations that include or exclude private methods, inner classes, enums, the standard library, or third-party libraries. That's $2^5 = 32$ variations. **DO NOT** create 32 variations for each metric. You'll increase the complexity of subsequent steps and eventually have to drop most variations due to multicollinearity problems anyway. **DO** include some variations that make sense theoretically. For example, when measuring coupling, including third-party libraries makes more sense than including the standard library [61].

Generating variations will not substantially improve content validity. Indeed, only one variation is likely to survive the subsequent weeding process. Including variations is more about improving the quality of the resulting instrument by ensuring the best variation of each metric is used.

At this stage you should include a metric even if you worry that it conflates two different constructs, or you're not sure which of your constructs it measures. Those issues will be resolved below.

6.2.4 Collect Data

We need to collect approximately ten observations per metric for a good factor analysis [18]. For example, if we have 152 class-level Java code metrics, we need about 1520 classes. We don't need a representative sample of all the world's Java classes—purposive sampling is fine at this stage [4]. It will be easier if all of the classes come from the same project. Not just any project will do, though. For example, if you have metrics that include enums, you need a project that has some enums. Once you've collected your sample of source code, calculate all the metrics.

6.2.5 Exploratory Factor Analysis

Factor analysis is a technique used to extract "factors" or "constructs" or "latent variables" from a set of "variables" or "reflective indicators" to reveal the underlying latent structure. Exploratory factor analysis (EFA) analyzes correlations between variables in a dataset to estimate which factor they would belong to. The higher the correlations among variables, the more likely they are to be measuring the same underlying factor. There are two kinds of factor analysis: exploratory and confirmatory. As the name suggests, an exploratory factor analysis (EFA) helps us explore which indicators reflect which factors, while confirmatory factor analysis (CFA) provides "confirmation" that the factors do represent the correct variables [18].

We can use EFA to explore the underlying factor structure of the data, that is, the degree to which metrics converge with other metrics for the same construct and diverge from metrics associated with other constructs. To reveal factors through EFA, we need many reflective indicators that ostensibly measure each construct and capture all aspects of each construct. We need lots of diverse metrics, not just a few metrics or a few variations on a single metric.

Guidelines for conducting an EFA are available elsewhere [18]. We include a worked example (Appendix 2) and provide a sample dataset and scripts that can be used for demonstrations or lab assignments (see Supplementary Materials on page 138). Needless to say, we do not simply compute an EFA and call it a day. We must iteratively refine the model until convergent validity and discriminant validity are high.

All of our example data is numeric. A factor analysis only accepts numerical data, so we'll have to recode any nonnumerical data to use it. We only consider unidimensional measures at the class level. Multidimensional measures require more sophisticated approaches (as discussed above); measures at different levels may require a multi-level modeling approach.

This exploratory part of the instrument development process includes many subjective decisions. Many of these decisions have no firm theoretical basis—we just have to make a reasonable decision and move on. Don't worry about p-hacking or over-fitting at this stage.

Once we're finished tweaking the EFA model, we should reassess content validity. That is, we should ensure that the remaining metrics cover all aspects of the constructs at hand. If so, the metrics included in the final EFA model are the only ones we need going forward. If not, we need to add some new metrics and try again.

6.2.6 Confirmatory Factor Analysis

At this point, we have a tentative measurement model, but it was devised subjectively, probably using data from a single project. Many of our choices cannot be justified beyond seeming reasonable at the time. So we want to confirm that our model holds up on a different and larger dataset.

Now sampling matters. We want a large sample of all the world's code that we can argue is representative (see [4]). Here, "large" could mean 7 projects or 7 thousand projects depending on the magnitude of manual steps and the time complexity of your analysis. (Note: we don't want snapshots of the same project across many different times—that may cause problems.) But we don't have a *sampling frame*: an index of all the world's code to sample from. So we have some options:

- Use a standard code corpus like Qualitas [60] or PyTorrent [3].[5] The advantage of a corpus is that our work is easier to replicate and compare to other studies using the same corpus. The disadvantage is that corpus is probably biased toward higher-quality code, while our measures should work (and be validated) on both good and bad code.
- Get codebase from a large company (or if the company can't share the code, get them company to run the analysis for you and send you the results). The problem here is every company is unique. We cannot generalize from Microsoft's code to Apple's or vice versa.
- Sample randomly from GitHub. This has the strongest argument to representativeness because of the diversity of projects hosted on GitHub at the time of writing. However, there is a possibility that open-source code systematically differs from closed-source code and that our study will be harder to replicate.
- Get two samples: one more diverse open-source sample and one less diverse closed-source sample from a partner organization. Do the CFA twice and compare the results. This is the most rigorous approach we can think of, but it's more laborious.

Exercises

Look at the list of things you want to measure. What kinds of organizations might care about measuring these things? Might any of them care enough to work with you? Do you know anyone at those organizations, or who could introduce you to people in those organizations?

Next, we calculate all of the metrics in our final EFA model and run CFA. For CFA, we must specify not only the number of factors but also which metrics correspond to which factors. Comprehensive guidelines for conducting CFA are available elsewhere [20], so here we will just include a few notes:

- Like EFA, CFA produces factor loadings that indicate how strongly each metric relates to its factor.
- Unlike EFA, CFA only produces loadings for the construct to which a metric is assigned.

[5] We have not used PyTorrent but it looks promising.

- EFA loadings range from −1 to 1. CFA loadings can be greater than 1, and that just means the variables loads highly on its corresponding factor.
- CFA produces estimates of each construct called "factor scores" that we can use for causal analysis.

CFA produces a new measurement model. It has the same metrics as the EFA model but a different formula for estimating the constructs from the metrics. These metrics and the formula produced by the CFA collectively make up our measurement model. Now we can use these formulas to estimate our constructs of interest without repeating the CFA (or any previous steps in the instrument development process).

During our EFA analysis, we can fiddle with the model as much as we want. But then we move from an exploratory phase to a confirmatory phase. This means we only run the CFA once, and then we write our instrument development paper and report our results. If some of our metrics don't load well, that means our measurement model is only "partially supported." In the discussion section of our paper, we can make recommendations like "consider dropping metric *m*," but we don't iterate at this stage because then it wouldn't be confirmatory testing anymore.

Later, when conducting a study *using* our instrument in causal analysis, we might find one or more metrics load poorly. *Then,* we can drop problematic metrics. Dropping problematic metrics is pretty common, which is one reason why it's useful to have extra metrics in our measurement model.

6.2.7 Additional Steps

By the end of the CFA process, we should be very confident in our measurement model. However, our confidence increases if our measure demonstrates measurement invariance (Sect. 3.2) and predictive validity. For example, if we had instruments for the size and complexity of a software system, and we found that systems became *less* complex as they grew, we would suspect that something was wrong with our measures.

However, we would argue that the time to write an instrument development paper is at the end of the CFA. Measurement invariance and predictive validity, in contrast, should be assessed when we use our new instruments to do some causal analysis. Therefore, further exploration of these topics is beyond the scope of this chapter.

6.3 Lessons Learned

Instrument development is easier to understand if you do it rather than just talking about it. For psychometric instruments, leading students through an elaborate simulation in which they actually create a questionnaire scale is an excellent way of conveying the process. This simulation must be performed over several classes because the instructor needs time for additional tasks, such as deduplicating the items.

The most difficult part is rating the favorability of each item toward the concept. For example, the item "My work makes me sad" has low favorability toward the concept "job satisfaction." Students who are not paying attention to the exercise sometimes rate it low because "My work makes me sad" sounds bad, or because *their* work doesn't make *them* sad. The instructor should go through some examples like this one to ensure the students understand the task.

As should be clear from the preceding guidelines, generating an instrument to measure a technical property of a software system is a lot to simulate in a single class. Instead, we recommend giving an overview of the process, followed by lab assignment corresponding to discrete steps, and the online supplement includes materials that can be used for an EFA demonstration or lab assignment.

7 Discussion and Conclusion

7.1 Summary

This chapter explores a broad spectrum of measurement issues in the context of software engineering research. Its core message is that software engineering research is being held back by a lack of attention to measurement. Notwithstanding some recent, excellent papers about measurement (e.g., [16, 50, 55]), much SE research is not trustworthy because it uses unreliable, unvalidated measures.

We began, in Sect. 2, by discussing the importance of selecting appropriate variables from the causal web surrounding one's phenomenon of interest. We argued that choosing variables that are too proximate reduces impact, while choosing variables too remote impedes research. Once we've selected variables, we must find ways of measuring them *reliably*. In Sect. 3, we argued for conceptualizing reliability in terms of True Score Theory, quantitatively assessing reliability and measurement invariance, and improving reliability through piloting, triangulation, and double-entering data.

Reliability is necessary but insufficient for validity. That is, you can consistently measure the wrong thing. Section 4 therefore introduced the realist view of *validity*—the degree to which a given instrument measures what it is supposed to measure. We argued that measurement is intrinsically problematic because observable changes in the world are often driven by unobservable structures and processes. This necessitates statistical measurement models, as explained in Sect. 5. We focused on reflective, common factor models, only touching upon the more difficult cases of multidimensional and causal-formative measurement models. The main takeaway of this section is that practically all technical properties of software systems and psychosocial properties of software developers(/teams/organizations) require sophisticated statistical measurement models and should not be operationalized as singular "proxy" variables. Section 6 therefore provides recommendations for building statistical measurement models for psychosocial and technical prop-

erties. Good instrumentation is arduous and, we argue, should constitute sufficient contribution for a full-length technical journal article.

7.2 Lessons Learned: Assessment Strategies

Assessing students' understanding of the concepts described above can be challenging. In some universities, research methods are taught in workshops, seminars, or lab meetings with little assessment or grading. Others have conventional courses with projects or exams. At the undergraduate level, exams can be used to assess superficial knowledge of measurement (e.g., "give examples of metrics"; "define systemic error"; "explain the importance of measurement invariance"). At the graduate level, however, exams cannot assess the ability to conceptualize latent structures, generate an appropriate measurement model, and quantitatively analyze construct validity.

Many graduate research methods courses end with writing a research proposal. In principle, working out the measurement strategy is a core part of a research proposal, so this kind of assignment should work. In practice, most students struggle to construct a research proposal in sufficient detail. For example, when proposing an experiment with human participants, students tend to produce hand-waving descriptions of procedures and materials rather than usable study protocols and task materials that participants could actually receive, understand, and complete. Students struggle to formulate appropriate statistical procedures including contingencies (e.g., "what will you do if the residuals are not normally distributed?").

We therefore have the following suggestions for alternative assessments. We would not expect an introductory research methods course to include all of these assessments; these are just possibilities.

- Write a detailed critique of a given instrument development paper.
- Create a rubric for evaluating instrument development papers similar to the SIGSOFT Empirical Standards [46].
- A series of labs and corresponding lab assignments in which students complete the steps in Sect. 6.2.
- *(Graduate level only)* Find a research paper with a comprehensive replication package including a complete dataset and a statistical measurement model (e.g., [47]), and replicate the analysis. Suggest (or, better yet, implement) specific improvements to the measurement model.

7.3 Conclusion

In conclusion, we stress the need for software engineering researchers to pay more attention to measurement, quantitatively assess reliability and validity, embrace sta-

tistical measurement models, and develop more high-quality scales and instruments. We call on reviewers not only to raise their expectations around measurement (e.g., when reviewing papers, proposals, and grant applications) but also to give more credit when measurement is done well and to recognize that creating and validating an instrument is a study on its own. Lastly, we call on educators who teach software engineering research to provide more explicit instruction around measurement issues, and we hope this chapter is a strong resource for helping in this regard.

Acknowledgments This work was supported by NSERC Discovery Grant RGPIN-2020-05001, Discovery Accelerator Supplement RGPAS-2020-00081, and the Izaak Walton Killam Postdoctoral fellowship program. Thanks are due to Klaas Stol and two anonymous reviews for their constructive feedback on this chapter.

Supplementary Materials

Supplementary materials for the reliability and exploratory factor analysis—including a dataset, sample scripts, sample results, and definitions of code quality metrics—can be found at https://doi.org/10.5281/zenodo.11544897.

Competing Interests The authors have no conflicts of interest to declare that are relevant to the content of this chapter.

Appendix 1: Reliability Analysis

This appendix provides an example of analyzing reliability on software metrics computed by different tools. The metrics were calculated from the source code of *Apache Maven*.[6] You can find the data and scripts in the online supplement to this book (see Supplementary Materials).

The purpose of this analysis is to investigate the extent to which metrics calculated by different tools provide consistent measurements.

Data Preparation

The dataset includes various metrics such as size, cohesion, inheritance, and coupling metrics, which are continuous. The following steps were undertaken to prepare the data for reliability analysis:

[6] https://github.com/apache/maven

1. Read the data from the Excel file containing the metrics.

```
library(readxl)
data <- read_excel("efaReadyMC.xlsx")
```

2. Select the relevant metrics for analysis. Here we use LOC metrics computed by three different tools—Designite,[7] JHawk,[8] and Understand.[9] The corresponding columns in the dataset are Size.LOC.Designite, Size.LOC.JHawk, and Size.LOC.Understand.

```
rel1_data <-
    select(rel_data, 'Size.LOC.Designite',
    'Size.LOC.JHawk', 'Size.LOC.Understand')
```

Calculate a Measure of Reliability

Since lines of code is ratio-level data, we'll use a measure of reliability rather than agreement (see Sect. 3.3.1). For this example, we will use Cronbach's alpha because it is simpler to calculate and interpret. Note, however, that if we were looking at reliability after doing factor analysis or a similar technique, more accurate measures of reliability such as McDonald's omega and composite reliability are available.

We will calculate Cronbach's alpha using the `psych` package[10] in R as follows:

1. Convert the selected data into a dataframe.

```
rel1_data <- as.data.frame(t(rel1_data))
```

2. Calculate alpha.

```
library(psych)
alphaResult <- alpha(rel1_data)
```

Results and Interpretation

Cronbach's alpha values range from -1 to 1, with higher values indicating greater reliability and internal consistency among the measured items. Generally, $\alpha > 0.7$ is considered acceptable, while $\alpha > 0.9$ is considered excellent. Our result $\alpha = 0.97$

[7] https://www.designite-tools.com/

[8] http://www.virtualmachinery.com/jhdownload.htm

[9] https://scitools.com/

[10] https://personality-project.org/r/psych/

indicates excellent reliability. This suggests that the three tested tools are measuring basically, if not exactly, the same thing. However, excellent reliability doesn't mean that the studied metrics reflect the target underlying construct (e.g., class size). To determine that, we need a different kind of analysis (next).

Appendix 2: Exploratory Factor Analysis

This appendix provides an example of an exploratory factor analysis, following established guidelines [18] and using selected metrics calculated from the source code of *Apache Maven*.[11] You can find the data and scripts in the online supplement to this book (see Supplementary Materials on page 138).

Objective of Factor Analysis

The objective of our exploratory factor analysis is to assess the convergent and discriminant validity of common, object-oriented, class-level software code quality metrics calculated on the source code of Apache Maven where convergent validity refers to how similar the measure is with other measures it should be theoretically similar to and discriminant validity refers to how different the measure is with other measures it should theoretically be different to [62].

Design the Factor Analysis

The dataset we will use contains measurements from 22 metrics from approximately 1000 classes, well over the 10 observations per variable threshold. We aim to classify these metrics into six factors:

1. Cohesion: the degree to which elements of a class belong together.
2. In-coupling: the degree to which a class is used by other classes.
3. Out-coupling: the degree to which a class depends on other classes.
4. Size: how big the class is.
5. Sub-inheritance: the degree to which a class has subclasses in an inheritance hierarchy.
6. Sup-inheritance: the degree to which a class has superclasses in an inheritance hierarchy.

[11] https://github.com/apache/maven

Check Assumptions of Factor Analysis

The assumptions of a factor analysis, and how we justify or test them, are as follows:

- Factor analysis should only be used when we theorize that a latent factor structure exists. In this case, we theorize that specific factors (size, coupling, etc.) are latent and do drive changes in metrics.
- Homogeneous sample of measurements. In other words metrics are calculated on the same sample of classes.
- Multicollinearity. If none of the variables are correlated, we cannot perform factor analysis; however, if two or more variables are perfectly or near perfectly correlated, it will cause a "nonpositive definite" matrix, which will prevent the factor analysis from completing. We can assess multicollinearity in three ways:

  1. Visually inspecting a correlation matrix. In this case, we can see many correlations > 0.3, which indicates that a factor analysis is possible [18].
  2. The Kaiser-Meyer-Olkin (KMO) test. The KMO test tells us how correlated the variables in a dataset are. A minimum KMO value of 0.5 is acceptable and a value above 0.7 is recommended for a good factor analysis [31]. Our $KMO = 0.71$ is considered "middling" and appropriate for factor analysis [31]. The KMO values for each individual metric are also greater than the acceptable minimum of 0.5 [31] (Fig. 8).
  3. Bartlett's test of sphericity. Bartlett's test of sphericity analyzes the correlations between variables to see if they are large enough to perform a factor analysis [14]. It was found that Bartlett's test of sphericity is significant ($p < 0.001$) and we can proceed with the factor analysis (Fig. 9).

Kaiser-Meyer-Olkin factor adequacy
Call: $KMO(r = efadataset)$
Overall MSA = 0.71
MSA for each item =

Size.NOM	0.72	Cohesion.YALCOM	0.88
Size.NOPM	0.75	Cohesion.LCOM	0.77
Size.LOC	0.79	Cohesion.LCOMModified	0.74
Size.CountInstanceVariable	0.88	Cohesion.LCOM5	0.59
Size.CountDeclMethodDefault	0.62	In-Coupling.FANINa	0.94
Sub-Inheritance.NC	0.61	In-Coupling.FANINb	0.54
Sup-Inheritance.DIT	0.65	Out-Coupling.FANOUTa	0.98
Sup-Inheritance.ReuseRatio	0.68	Out-Coupling.FANOUTb	0.56
Sub-Inheritance.CountSub	0.56	Out-Coupling.CBOout	0.92
Sup-Inheritance.CountSup	0.61	In-Coupling.CBOin	0.62
Sub-Inheritance.SpecializationRatio	0.61		

Fig. 8 Kaiser-Meyer-Olkin (KMO) test

Bartlett's Test of Sphericity
Call: $bart_spher(x = efadataset)$
X2 = 35955.426
df = 210
$p < 0.001$

Fig. 9 Bartlett's test of sphericity

Derive Factors and Assess Fit

Researchers disagree on the best method of determining the number of factors to extract. We recommend using several methods to inform the decision [11].

Parallel analysis is a technique to estimate factors by calculating eigenvalues of random, uncorrelated data with eigenvalues of the actual data. The number of factors to retain is the count of eigenvalues greater than 0 [24]. In our data, 21 factors are retained—21 eigenvalues are greater than 0 (Fig. 10). Parallel analysis is known to overestimate the number of factors extracted from very large datasets like ours [11].

Alternatively, we retain a number of factors equal to the number of eigenvalues greater than 1 (the *Kaiser Criterion* [30]). This method also loses effectiveness as the size of the dataset increases [63], but not so much. We found five eigenvalues greater than 1 (Fig. 11), suggesting that we retain five factors.

We can also estimate the number of factors to retain by counting the eigenvalues before the bend in a *scree plot* [8]. This technique is a little tricky and requires expertise if the plot is complicated [11]. Figure 12 has multiple bends—at two, four, and seven eigenvalues—which suggests retaining anywhere between two and seven factors.

In the *theory* approach, we retain the number of factors that we theorize exist—in this case, six: size, cohesion, sub-inheritance, sup-inheritance, in-coupling, and out-coupling.

From the above discussion, we have the following findings:

1. Parallel analysis suggests 21 factors.
2. The Kaiser Criteria suggest five factors.
3. The scree plot suggests factors between 2 and 7.
4. Theory suggests six factors.

Based on this, we tentatively retain seven factors as shown in Table 1. Table 1 shows the variables and their corresponding factor loadings on each of the seven factors. Factor loading refers to the correlation between a variable and a factor. A high loading suggests that the variance explained by a variable is sufficient for it to have a considerable relationship with the factor. Small loadings (< 0.3) are

Using eigendecomposition of correlation matrix.
Computing: 10% 20% 30% 40% 50% 60% 70% 80% 90% 100%
Results of Horn's Parallel Analysis for factor retention
630 iterations, using the mean estimate

Factor	Adjusted Eigenvalue	Unadjusted Eigenvalue	Estimated Bias
No components passed			
1	5.514237	5.778228	0.263991
2	2.897594	3.120152	0.222557
3	2.456621	2.646264	0.189642
4	2.216814	2.37921	0.162396
5	1.501931	1.638783	0.136852
6	0.902975	1.01652	0.113544
7	0.67688	0.76922	0.09234
8	0.199953	0.270978	0.071025
9	0.13904	0.189688	0.050648
10	0.060382	0.090571	0.030188
11	0.041161	0.052015	0.010853
12	0.012305	0.004267	-0.00803
13	0.027372	0.000565	-0.0268
14	0.045743	-0.00027	-0.04602
15	0.057293	-0.00824	-0.06554
16	0.047827	-0.03736	-0.08518
17	0.060359	-0.0445	-0.10486
18	0.070699	-0.05492	-0.12562
19	0.089972	-0.05789	-0.14787
20	0.054164	-0.11756	-0.17172
21	0.023885	-0.17774	-0.20163

Adjusted eigenvalues > 0 indicate dimensions to retain.
(21 factors retained)

Fig. 10 Parallel analysis

considered insignificant [11, 18, 25, 51] and are thus suppressed in our model. For example, "Cohesion.LCOM, Cohesion.LCOMModified, Cohesion.YALCOM," and "Size.CountInstanceVariable" load together on Factor 1, which means that these variables seem to be measuring the same factor.

These factors explain 84% of the variance in our dataset, which is good. If the variance explained was less than 60%, we might opt to include additional factors [18]. However, Factor 5 only has two metrics loading on it. The minimum is three, so either we need more metrics or fewer factors.

Parallel analysis suggests that the number of factors = 7 and the number of components = NA
Call: $fa.parallel(x = efadataset, fm = "pa", fa = "fa")$
Parallel analysis suggests that the number of factors = 7 and the number of components = NA
Eigen Values of

eigen values of factors

[1]	5.35	2.45	1.83	1.53	1.16	0.63	0.26	-0.01	-0.25	-0.31
-0.37	-0.42	-0.51	-0.54	-0.59	-0.63	-0.67	-0.78	-0.89	-0.94	-0.95

eigen values of simulated factors

[1]	0.3	0.21	0.18	0.16	0.13	0.11	0.09	0.06	0.04	0.03
0.01	-0.01	-0.03	-0.05	-0.07	-0.08	-0.1	-0.13	-0.15	-0.18	-0.21

eigen values of components

[1]	5.93	3.27	2.73	2.49	1.77	1.28	1.07	0.57	0.46	0.35
0.3	0.22	0.15	0.1	0.08	0.06	0.06	0.04	0.03	0.03	0

eigen values of simulated components

[1]	NA

Fig. 11 Kaiser Criterion

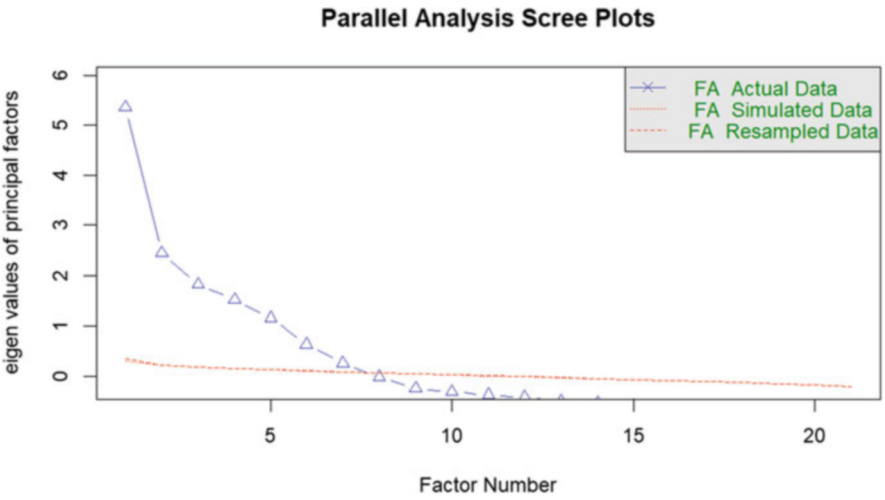


Fig. 12 Scree plot

Table 1 EFA with seven factors

	F1	F2	F3	F4	F5	F6	F7	h2
Cohesion.LCOM	0.95							0.92
Cohesion.LCOM5					0.64			0.4
Cohesion.LCOMModified	0.98							0.93
Cohesion.YALCOM	0.63							0.45
In-Coupling.CBOin		0.87	0.36					1
In-Coupling.FANINa		0.93						0.94
In-Coupling.FANINb		0.99						1
Out-Coupling.CBOout			0.8					0.81
Out-Coupling.FANOUTa			0.92					0.9
Out-Coupling.FANOUTb			0.98					0.95
Size.CountDeclMethodDefault					0.92			0.9
Size.CountInstanceVariable	0.44			0.33				0.59
Size.LOC			0.31	0.71				0.69
Size.NOM.Designite				0.97				1
Size.NOPM.Understand				0.71				0.73
Sub-Inheritance.CountSub						1		1
Sub-Inheritance.NC						0.73		0.62
Sub-Inheritance.SpecializationRatio						0.94		0.9
Sup-Inheritance.CountSup							0.98	0.96
Sup-Inheritance.DIT							0.9	0.86
Sup-Inheritance.ReuseRatio							0.96	0.92

In this case, we can reduce the number of factors to six, as theorized. Table 2 shows the six-factor solution. This solution explains 78% of the variance and each factor has at least three metrics, so we can move on to iteratively refining and interpreting the factors.

(We included this step to illustrate the realistic complexity of choosing an appropriate number of factors. Sometimes you're well into the analysis before you figure out how many factors you should have.)

Interpret and Refine Factors

Rotating the factors helps us interpret them. In a rotated factor solution, the axes are rotated so that variables that load together are plotted closer together causing them to load highly on a single factor.

Table 2 EFA with six factors (step 1)

	Cohesion	InCoupling	OutCoupling	Size	SubInheritance	SupInheritance	h2
Cohesion.LCOM	0.93						0.92
Cohesion.LCOM5		**0.42**					**0.16**
Cohesion.LCOMModified	0.98						0.94
Cohesion.YALCOM	0.62						0.45
In-Coupling.CBOin		0.84	0.35				0.95
In-Coupling.FANINa		0.94					0.92
In-Coupling.FANINb		0.96					0.93
Out-Coupling.CBOout			0.8				0.81
Out-Coupling.FANOUTa			0.93				0.9
Out-Coupling.FANOUTb			0.98				0.95
Size.CountDeclMethodDefault		0.53					0.32
Size.CountInstanceVariable	0.42			0.34			0.59
Size.LOC			0.3	0.74			0.71
Size.NOM.Designite				0.9			0.89
Size.NOPM.Understand				0.76			0.71
Sub-Inheritance.CountSub					1		1
Sub-Inheritance.NC					0.73		0.63
Sub-Inheritance.SpecializationRatio					0.94		0.91
Sup-Inheritance.CountSup						0.98	0.96
Sup-Inheritance.DIT						0.91	0.87
Sup-Inheritance.ReuseRatio						0.95	0.91

Factors can be rotated using orthogonal or oblique rotation. An oblique rotation is preferable when factors are assumed to be correlated; an orthogonal rotation is used otherwise [11, 14, 18]. Despite the popularity of orthogonal rotation (specifically varimax), oblique rotation is usually more appropriate because factors are usually correlated. Use orthogonal rotation only if you have a very good reason to believe that the factors are uncorrelated. We use *oblimin rotation* (a type of oblique rotation) to rotate the axis in our model.

Now we inspect the solution for problems, and remove problems one at a time, beginning with the worst. There is no algorithm for this. "Worst" is subjective. We can only give examples of problems and describe their severity. We are looking for three basic kinds of problems:

1. Low communality: Communality ("h2" in Tables 1 and 2) is the amount of variance in a variable that can be explained by the factor solution. Low communality ($h2 < 0.5$) indicates that less than half of the variance of the variable is taken into account implying that the variable is not closely related to any of the factors and causes unwanted complexity with insufficient explanation [18].
2. Cross-loadings: Variables with high loadings on multiple factors.
3. Loading on the wrong factor: Variables loading highly (loadings > 0.5) on a factor they shouldn't be.

Looking at Table 2, *Cohesion.LCOM5* has the lowest communality ($h2 = 0.16$) *and* loads on the wrong factor (in-coupling), so we remove that one first and rerun the EFA (Table 3). Now *Size.CountDeclMethodDefault* has the lowest communality ($h2 = 0.25$) and loads on the wrong factor (in-coupling again). So we remove it and run the EFA again (Table 4). The next variable with lowest communality is *Cohesion.YALCOM* ($h2 = 0.45$); however, it loads well on the correct factor, so we'll retain it for now and move onto cross-loadings. *Size.CountInstanceVariable* loads higher on the wrong factor (cohesion) than on the correct factor (size). Thus, we remove it. Rerunning the EFA (Table 5), we find that *In-Coupling.CBOin* also has a cross-loading. However, it loads much higher on the correct factor (in-coupling) than the incorrect factor (out-coupling). Furthermore, the incorrect loading is smaller than the smallest correct loading in the EFA, so we retain *Size.CountInstanceVariable* for now.

Since there are no more low communalities, cross-loadings, or incorrect loadings, and the solution explains 87% of the total variance in the dataset using six factors, each of which has at least three metrics, our EFA is now complete (Table 6).

Table 3 EFA with six factors (step 2)

	Cohesion	InCoupling	OutCoupling	Size	SubInheritance	SupInheritance	h2
Cohesion.LCOM	0.93						0.92
Cohesion.LCOMModified	0.97						0.93
Cohesion.YALCOM	0.62						0.45
In-Coupling.CBOin		0.87	0.36				0.99
In-Coupling.FANINa		0.96					0.95
In-Coupling.FANINb		0.99					0.98
Out-Coupling.CBOout			0.8				0.81
Out-Coupling.FANOUTa			0.92				0.9
Out-Coupling.FANOUTb			0.98				0.95
Size.CountDeclMethodDefault		**0.43**					**0.25**
Size.CountInstanceVariable	0.43			0.35			0.59
Size.LOC				0.73			0.7
Size.NOM				0.96			0.96
Size.NOPM				0.71			0.67
Sub-Inheritance.CountSub					1		1
Sub-Inheritance.NC					0.73		0.62
Sub-Inheritance.SpecializationRatio					0.94		0.9
Sup-Inheritance.CountSup						0.98	0.96
Sup-Inheritance.DIT						0.91	0.87
Sup-Inheritance.ReuseRatio						0.95	0.91

Table 4 EFA with six factors (step 3)

	Cohesion	InCoupling	OutCoupling	Size	SubInheritance	SupInheritance	h2
Cohesion.LCOM	0.94						0.92
Cohesion.LCOMModified	0.98						0.95
Cohesion.YALCOM	0.62						0.45
In-Coupling.CBOin		0.87	0.36				1
In-Coupling.FANINa		0.95					0.93
In-Coupling.FANINb		1					1
Out-Coupling.CBOout			0.8				0.81
Out-Coupling.FANOUTa			0.92				0.9
Out-Coupling.FANOUTb			0.98				0.95
Size.CountInstanceVariable	**0.42**			**0.35**			**0.59**
Size.LOC				0.74			0.7
Size.NOM.Designite				0.95			0.93
Size.NOPM.Understand				0.71			0.67
Sub-Inheritance.CountSub					1		1
Sub-Inheritance.NC					0.73		0.62
Sub-Inheritance.SpecializationRatio					0.94		0.9
Sup-Inheritance.CountSup						0.98	0.96
Sup-Inheritance.DIT						0.91	0.86
Sup-Inheritance.ReuseRatio						0.96	0.92

Table 5 EFA with six factors (step 4)

	Cohesion	InCoupling	OutCoupling	Size	SubInheritance	SupInheritance	h2
Cohesion.LCOM	0.95						0.95
Cohesion.LCOMModified	0.96						0.91
Cohesion.YALCOM	0.64						0.47
In-Coupling.CBOin		**0.87**	**0.36**				**1**
In-Coupling.FANINa		0.95					0.94
In-Coupling.FANINb		1					1
Out-Coupling.CBOout			0.8				0.81
Out-Coupling.FANOUTa			0.92				0.9
Out-Coupling.FANOUTb			0.98				0.95
Size.LOC				0.74			0.72
Size.NOM.Designite				0.97			0.98
Size.NOPM.Understand				0.67			0.61
Sub-Inheritance.CountSub					1		1
Sub-Inheritance.NC					0.73		0.62
Sub-Inheritance.SpecializationRatio					0.94		0.9
Sup-Inheritance.CountSup						0.98	0.97
Sup-Inheritance.DIT						0.91	0.86
Sup-Inheritance.ReuseRatio						0.96	0.92

Table 6 Final EFA

	Cohesion	InCoupling	OutCoupling	Size	SubInheritance	SupInheritance	h2
Cohesion.LCOM	0.95						0.95
Cohesion.LCOMModified	0.96						0.91
Cohesion.YALCOM	0.64						0.47
In-Coupling.CBOin		0.87					1
In-Coupling.FANINa		0.95					0.94
In-Coupling.FANINb		1					1
Out-Coupling.CBOout			0.8				0.81
Out-Coupling.FANOUTa			0.92				0.9
Out-Coupling.FANOUTb			0.98				0.95
Size.LOC				0.74			0.72
Size.NOM.Designite				0.97			0.98
Size.NOPM.Understand				0.67			0.61
Sub-Inheritance.CountSub					1		1
Sub-Inheritance.NC					0.73		0.62
Sub-Inheritance.SpecializationRatio					0.94		0.9
Sup-Inheritance.CountSup						0.98	0.97
Sup-Inheritance.DIT						0.91	0.86
Sup-Inheritance.ReuseRatio						0.96	0.92

References

1. Agbo, A.A.: Cronbach's alpha: review of limitations and associated recommendations. J. Psychol. Africa **20**(2), 233–239 (2010)
2. Archer, M., Bhaskar, R., Collier, A., Lawson, T., Norrie, A.: Critical Realism: Essential Readings. Routledge, London (2013)
3. Bahrami, M., Shrikanth, N.C., Ruangwan, S., Liu, L., Mizobuchi, Y., Fukuyori, M., Chen, W.P., Munakata, K., Menzies, T.: Pytorrent: A python library corpus for large-scale language models (2021). arXiv [cs.SE]. https://doi.org/10.48550/arXiv.2110.01710
4. Baltes, S., Ralph, P.: Sampling in software engineering research: a critical review and guidelines. Empir. Softw. Eng. **27**(4), 94 (2022)
5. Basilevsky, A.: Statistical Factor Analysis and Related Methods: Theory and Applications. Wiley Series in Probability and Statistics. Wiley, London (1994)
6. Briggs, D.: Historical and Conceptual Foundations of Measurement in the Human Sciences: Credos and Controversies. Routledge, London (2021)
7. Campbell, N.: Physics: The Elements. Cambridge University Press, Cambridge (2013)
8. Cattell, R.B.: The scree test for the number of factors. Multivariate Behav. Res. **1**(2), 245–276 (1966)
9. Cerri, L.Q., Justo, M.C., Clemente, V., Gomes, A.A., Pereira, A.S., Marques, D.R.: Insomnia severity index: a reliability generalisation meta-analysis. J. Sleep Res. **32**(4), e13835 (2023)
10. Coltman, T., Devinney, T.M., Midgley, D.F., Venaik, S.: Formative versus reflective measurement models: two applications of formative measurement. J. Bus. Res. **61**(12), 1250–1262 (2008)
11. Costello, A., Osborne, J.: Best practices in exploratory factor analysis: four recommendations for getting the most from your analysis. Pract. Assessment Res. Eval. **10**, 1–9 (2005)
12. Drost, E.A.: Validity and reliability in social science research. Educ. Res. Perspect. **38**(1), 105–123 (2011)
13. Fassott, G., Henseler, J.: Formative (measurement). In: Wiley Encyclopedia of Management. John Wiley and Sons, London (2015). https://doi.org/10.1002/9781118785317.weom090113
14. Field, A.: Discovering Statistics Using IBM SPSS Statistics, 5th edn. Sage (2017)
15. Flater, D.W., Black, P.E., Fong, E.N., Kacker, R.N., Okun, V., Wood, S.S., Kuhn, D.R.: A rational foundation for software metrology. Tech. Rep. IR 8101, National Institute of Standards and Technology (2016). https://doi.org/10.6028/NIST.IR.8101
16. Graziotin, D., Lenberg, P., Feldt, R., Wagner, S.: Psychometrics in behavioral software engineering: a methodological introduction with guidelines. ACM Trans. Softw. Eng. Methodol. **31**(1), 1–36 (2021)
17. Gwet, K.L.: Handbook of Inter-Rater Reliability: The Definitive Guide to Measuring the Extent of Agreement Among Raters. Advanced Analytics, LLC (2014)
18. Hair, J., Black, W., Babin, B., Anderson, R.: Multivariate Data Analysis. Always Learning. Pearson Education Limited (2013)
19. Hair, J.F., Risher, J.J., Sarstedt, M., Ringle, C.M.: When to use and how to report the results of PLS-SEM. Eur. Bus. Rev. **31**(1), 2–24 (2019)
20. Harrington, D.: Confirmatory Factor Analysis. Oxford University Press, Oxford (2009)
21. Heilmann, C.: A new interpretation of the representational theory of measurement. Philos. Sci. **82**, 787–797 (2015)
22. Henseler, J.: Composite-Based Structural Equation Modeling. The Guilford Press (2021)
23. Herzig, K., Just, S., Zeller, A.: It's not a bug, it's a feature: how misclassification impacts bug prediction. In: 2013 35th International Conference on Software Engineering (ICSE), pp. 392–401 (2013). https://doi.org/10.1109/ICSE.2013.6606585
24. Horn, J.: A rationale and test for the number of factors in factor analysis. Psychometrika **30**, 179–185 (1965). https://doi.org/10.1007/BF02289447

25. Howard, M.: A review of exploratory factor analysis (EFA) decisions and overview of current practices: What we are doing and how can we improve? Int. J. Hum.-Comput. Interact. **32**, 150914142834000 (2015). https://doi.org/10.1080/10447318.2015.1087664
26. Hume, D.: A Treatise of Human Nature. Oxford University Press, Oxford (1896)
27. ISO/IEC/IEEE International Standard – Systems and Software Engineering–Vocabulary. Standard, IEEE, Switzerland. https://doi.org/10.1109/IEEESTD.2017.8016712
28. Johnson, P., Ekstedt, M., Jacobson, I.: Where's the theory for software engineering? IEEE Softw. **29**(5), 96–96 (2012)
29. Johnston, R.B., Smith, S.P.: How critical realism clarifies validity issues in theory-testing research: analysis and case. In: Information Systems Foundations: The Role of Design Science, pp. 21–48. ANU Press (2010)
30. Kaiser, H.F.: The application of electronic computers to factor analysis. Educ. Psychol. Measur. **20**(1), 141–151 (1960). https://doi.org/10.1177/001316446002000116
31. Kaiser, H.F., Rice, J.: Little jiffy, mark IV. Educ. Psychol. Measur. **34**(1), 111–117 (1974). https://doi.org/10.1177/001316447403400115
32. Kimberlin, C.L., Winterstein, A.G.: Validity and reliability of measurement instruments used in research. Am. J. Health-Syst. Pharmacy **65**(23), 2276–2284 (2008)
33. Krippendorff, K.: Content Analysis: An Introduction to its Methodology. Sage (2018)
34. Kuhn, T.S.: The function of measurement in modern physical science. Isis **52**(2), 161–193 (1961)
35. Liu, Y., Schuberth, F., Liu, Y., Henseler, J.: Modeling and assessing forged concepts in tourism and hospitality using confirmatory composite analysis. J. Bus. Res. **152**, 221–230 (2022). https://doi.org/10.1016/j.jbusres.2022.07.040
36. Macdonald, S., Maclntyre, P.: The generic job satisfaction scale: Scale development and its correlates. Empl. Assist. Q. **13**(2), 1–16 (1997)
37. Martín-Escudero, P., Cabanas, A.M., Dotor-Castilla, M.L., Galindo-Canales, M., Miguel-Tobal, F., Fernández-Pérez, C., Fuentes-Ferrer, M., Giannetti, R.: Are activity wrist-worn devices accurate for determining heart rate during intense exercise? Bioengineering **10**(2), 254 (2023)
38. McGuire, S., Schultz, E., Ayoola, B., Ralph, P.: Sustainability is stratified: toward a better theory of sustainable software engineering. In: 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE), pp. 1996–2008 (2023). https://doi.org/10.1109/ICSE48619.2023.00169
39. Michell, J.: Measurement in Psychology: A Critical History of a Methodological Concept, vol. 53. Cambridge University Press, Cambridge (1999)
40. Mohanani, R., Ralph, P., Turhan, B., Mandić, V.: How templated requirements specifications inhibit creativity in software engineering. IEEE Trans. Softw. Eng. **48**(10), 4074–4086 (2022). https://doi.org/10.1109/TSE.2021.3112503
41. Mohanani, R., Turhan, B., Ralph, P.: Requirements framing affects design creativity. IEEE Trans. Softw. Eng. **47**(5), 936–947 (2021). https://doi.org/10.1109/TSE.2019.2909033
42. Parry, O., Kapfhammer, G.M., Hilton, M., McMinn, P.: A survey of flaky tests. ACM Trans. Softw. Eng. Methodol. **31**(1) (2021). https://doi.org/10.1145/3476105
43. Passmore, J.: Logical positivism. In: Edwards, P. (ed.) the Encyclopedia of Philosophy, vol. 5, pp. 52–57. Macmillan, New York (1967)
44. Petter, S., Straub, D., Rai, A.: Specifying formative constructs in information systems research. MIS Quarterly 623–656 (2007)
45. Putnick, D.L., Bornstein, M.H.: Measurement invariance conventions and reporting: the state of the art and future directions for psychological research. Dev. Rev. **41**, 71–90 (2016)
46. Ralph, P., bin Ali, N., Baltes, S., Bianculli, D., Diaz, J., Dittrich, Y., Ernst, N., Felderer, M., Feldt, R., Filieri, A., de França, B.B.N., Furia, C.A., Gay, G., Gold, N., Graziotin, D., He, P., Hoda, R., Juristo, N., Kitchenham, B., Lenarduzzi, V., Martínez, J., Melegati, J., Mendez, D., Menzies, T., Molleri, J., Pfahl, D., Robbes, R., Russo, D., Saarimäki, N., Sarro, F., Taibi, D., Siegmund, J., Spinellis, D., Staron, M., Stol, K., Storey, M.A., Taibi,

D., Tamburri, D., Torchiano, M., Treude, C., Turhan, B., Wang, X., Vegas, S.: Empirical standards for software engineering research (2021). arXiv [cs.SE]. https://doi.org/10.48550/arXiv.2010.03525
47. Ralph, P., Baltes, S., Adisaputri, G., Torkar, R., Kovalenko, V., Kalinowski, M., Novielli, N., Yoo, S., Devroey, X., Tan, X., et al.: Pandemic programming: How COVID-19 affects software developers and how their organizations can help. Empir. Softw. Eng. **25**, 4927–4961 (2020)
48. Ralph, P., Kelly, P.: The dimensions of software engineering success. In: Proceedings of the 36th International Conference on Software Engineering, ICSE 2014, pp. 24–35. Association for Computing Machinery, New York (2014). https://doi.org/10.1145/2568225.2568261
49. Ralph, P., Tempero, E.: Construct validity in software engineering research and software metrics. In: Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018, pp. 13–23 (2018)
50. Russo, D., Stol, K.J.: PLS-SEM for software engineering research: an introduction and survey. ACM Comput. Surv. **54**(4), 1–38 (2021)
51. Samuels, P.C.: Advice on exploratory factor analysis. Tech. rep., Birmingham City University (2017). https://api.semanticscholar.org/CorpusID:201395127
52. Santos, R.D.S., Ralph, P., Arshad, A., Stol, K.J.: Distributed scrum: a case meta-analysis. ACM Comput. Surv. **56**(4) (2023). https://doi.org/10.1145/3626519
53. Sayer, A.: Method in Social Science, Revised 2nd edn. Routledge, London (2010)
54. Scott, H., Havercamp, S.M.: Measurement error. In: Volkmar, F.R. (ed.) Encyclopedia of Autism Spectrum Disorders, pp. 1817–1818. Springer, New York (2013)
55. Sjøberg, D.I., Bergersen, G.R.: Construct validity in software engineering. IEEE Trans. Softw. Eng. **49**(3), 1374–1396 (2022)
56. Stol, K.J., Fitzgerald, B.: Theory-oriented software engineering. Sci. Comput. Program. **101**, 79–98 (2015)
57. Tal, E.: Old and new problems in philosophy of measurement. Philo. Comp. **8**(12), 1159–1173 (2013)
58. Tal, E.: Measurement in Science. In: Zalta, E.N. (ed.) The Stanford Encyclopedia of Philosophy, Fall 2020 edn. Metaphysics Research Lab, Stanford University (2020). https://plato.stanford.edu/archives/fall2020/entries/measurement-science/
59. Tavakol, M., Dennick, R.: Making sense of Cronbach's alpha. Int. J. Med. Educ. **2**, 53 (2011)
60. Tempero, E., Anslow, C., Dietrich, J., Han, T., Li, J., Lumpe, M., Melton, H., Noble, J.: The Qualitas corpus: a curated collection of Java code for empirical studies. In: 2010 Asia Pacific Software Engineering Conference, pp. 336–345. IEEE, Piscataway (2010)
61. Tempero, E., Ralph, P.: A framework for defining coupling metrics. Sci. Comput. Program. **166**, 214–230 (2018). https://doi.org/10.1016/j.scico.2018.02.004
62. Trochim, W., Donnelly, J.P., Arora, K.: Research Methods: The Essential Knowledge Base, 2nd edn. Cengage Learning, Boston (2016)
63. Velicer, W.F., Eaton, C.A., Fava, J.L.: Construct explication through factor or component analysis: a review and evaluation of alternative procedures for determining the number of factors or components. In: Goffin, R.D., Helmes, E. (eds.) Problems and Solutions in Human Assessment, pp. 41–71. Springer, Boston (2000)
64. Ward, Z.B.: On Value-Laden Science. Studies in History and Philosophy of Science Part A, vol. 85, pp. 54–62 (2021)
65. Zinbarg, R.E., Revelle, W., Yovel, I., Li, W.: Cronbach's α, Revelle's β, and McDonald's ω_H: their relations with each other and two alternative conceptualizations of reliability. Psychometrika **70**, 123–133 (2005)

Qualitative Data Analysis in Software Engineering: Techniques and Teaching Insights

Christoph Treude

Abstract Software repositories are rich sources of qualitative artifacts, including source code comments, commit messages, issue descriptions, and documentation. These artifacts offer many interesting insights when analyzed through quantitative methods, as outlined in the chapter on mining software repositories. This chapter shifts the focus toward interpreting these artifacts using various qualitative data analysis techniques. We introduce qualitative coding as an iterative process, which is crucial not only for educational purposes but also to enhance the credibility and depth of research findings. Various coding methods are discussed along with the strategic design of a coding guide to ensure consistency and accuracy in data interpretation. The chapter also discusses quality assurance in qualitative data analysis, emphasizing principles such as credibility, transferability, dependability, and confirmability. These principles are vital to ensure that the findings are robust and can be generalized in different contexts. By sharing best practices and lessons learned, we aim to equip all readers with the tools necessary to conduct rigorous qualitative research in the field of software engineering.

1 Introduction

In the area of software development, understanding and analyzing qualitative data from software repositories can produce many valuable insights. These repositories are rich with insightful artifacts, such as source code comments, commit messages, issue descriptions, and documentation [26]. Following the mining processes detailed in a previous chapter, this chapter focuses on how to interpret and analyze these artifacts using qualitative data analysis methods, with a special emphasis on teaching these methods.

Qualitative data analysis extends beyond mere data collection; it involves a thorough and iterative process of coding and interpretation [43]. This chapter

C. Treude (✉)
Singapore Management University, Singapore, Singapore
e-mail: ctreude@smu.edu.sg

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2024

D. Mendez et al. (eds.), *Handbook on Teaching Empirical Software Engineering*,
https://doi.org/10.1007/978-3-031-71769-7_6

introduces qualitative coding not just as a technique but as a systematic process integral to educational practice in software engineering. We explore various coding methods and emphasize the importance of designing a robust coding guide [42], tailored for researchers and educators.

We target an audience that includes students, researchers, software developers, and especially educators, providing them with the necessary tools and insights to effectively analyze qualitative data and teach these skills. Furthermore, the chapter stresses the importance of quality assurance in qualitative data analysis, highlighting key principles such as credibility, transferability, dependability, and confirmability [31] to ensure that the analysis is not only thorough but also sound. We also address common challenges and pitfalls in qualitative data analysis, offering practical advice and best practices specifically tailored for educational settings [9]. Lastly, the chapter discusses the nuances of teaching qualitative data analysis in software engineering, sharing lessons learned, and effective teaching practices.

1.1 Target Audience

This chapter is particularly designed for educators in software engineering, providing them with a comprehensive guide to teaching qualitative data analysis. The reader will understand the significance of qualitative data in software engineering and will explore various types of software engineering artifact that can be analyzed using qualitative coding. The chapter offers a detailed description of the qualitative coding process and includes instructions on designing an effective coding guide. It emphasizes the importance of quality assurance in qualitative data analysis, providing checklists to ensure rigor and reliability. In addition, common mistakes and best practices are discussed to improve educators' ability to instruct effectively. Educators will not only gain insight into applying these methods but also acquire practical tools to inspire and engage their students in adopting these techniques in their own projects.

1.2 Learning Objectives

Upon completing this chapter, readers should be able to:

- Identify and understand different qualitative artifacts commonly found in software engineering
- Apply systematic coding techniques to qualitative data to derive meaningful insights, suitable for classroom instruction
- Design and implement a coding guide tailored to both educational purposes and specific research needs

- Recognize and ensure the quality aspects of qualitative analysis, such as credibility and dependability, essential for teaching sound research methods
- Navigate common challenges in qualitative data analysis and employ best practices to overcome them

1.3 Teaching Strategies

To effectively teach the content of this chapter, the following exemplary strategies are recommended for educators:

Interactive Workshops: Facilitate interactive coding sessions in classrooms where students can practice coding real data extracted from software repositories. This hands-on approach helps cementing theoretical knowledge through practical application.

Case Studies: Use case studies from published papers to illustrate the application and results of qualitative coding. These examples can serve as a discussion base for understanding complex concepts and their implications in real-world settings.

Peer Review: Encourage students to peer-review each other's coding projects to foster a collaborative learning environment and critical thinking skills. This method also improves their ability to constructively critique and improve qualitative analyses.

Role-Playing: Implement role-playing exercises where students assume various roles involved in qualitative data analysis, such as data collector, coder, and quality assurer. This strategy helps students understand the collaborative and interdisciplinary nature of software engineering projects.

2 Qualitative Artifacts in Software Repositories

Software repositories are not just codebases; they are intricate ecosystems full of valuable information. Over the past decades, software analytics and mining software repositories have evolved into significant fields of research and practice [22]. These activities involve sifting through large amounts of data, uncovering patterns, and extracting actionable insights to improve the software development process.

Although quantitative analysis has been the traditional focus, offering valuable insights, such as using metrics such as code complexity or line counts to predict defects [5], there is growing recognition of the untapped potential in qualitative data. Qualitative artifacts, such as source code comments, commit messages, bug reports, developer conversations, interactions with language learning models (LLMs), and documentation, hold a wealth of information that can provide a deeper contextual understanding of the software development process. Each of these artifacts brings a different lens to the analysis, contributing to a multifaceted understanding of both

technical and human factors in software projects. The following are a few concrete examples.

Source Code Comments and Commit Messages Source code comments and commit messages are often overlooked sources of information. Qualitative analysis here can uncover the narrative behind complex code changes [7]. For instance, a series of commit messages might reveal a developer's struggle with a particular bug, showcasing the iterative process of trial and error before the final solution. Comments in the code might also highlight areas where developers felt uncertain or where they anticipated future changes, indicating potential hotspots for future errors or refactoring needs. This analysis can guide project managers and team leaders in understanding team challenges, leading to targeted interventions or knowledge sharing sessions. The primary challenge in qualitatively analyzing these artifacts lies in their inconsistency and brevity. Developers may not consistently comment on their code or may write commit messages that are cryptic or too concise, making it difficult to derive meaningful insights without extensive contextual knowledge. For example, a cryptic commit message like "big change" appears frequently in repositories and often lacks a detailed explanation, complicating the task of tracing specific changes and their rationales. At the time of writing, a GitHub search for commit messages containing "big change" returned more than a million hits: https://github.com/search?q=%22big+change%22&type=commits.

Bug Reports Bug reports, when qualitatively analyzed, can offer much more than the identification of glitches [4]. They can serve as a narrative of the user's journey, pinpointing not just technical failures but also misalignments between user expectations and software functionality. For example, a consistent theme in bug reports might not indicate a code defect but rather a feature that is counterintuitive to users, necessitating a redesign rather than a simple fix. Furthermore, the language used in these reports can indicate the severity and emotional impact of bugs on users, providing a measure of user satisfaction and urgency for fixes. The challenge in analyzing bug reports qualitatively is their sheer volume and variability. Each report may vary greatly in detail and quality, which requires researchers to sift through potentially hundreds of reports to find relevant data. As an illustrative point, searches for vague complaints like "nothing works" currently yield more than 100,000 results on GitHub (https://github.com/search?q=%22nothing+works%22&type=issues), each potentially varying greatly in context and severity, thus complicating the coding process.

Developer Conversations Developer conversations, found in forums, pull requests, or even chat logs, are rich in insight into team dynamics and decision-making processes [23, 48]. Qualitative analysis of these conversations can highlight how decisions are made, whether there is a dominant voice that guides decision-making or whether there is a healthy debate before reaching a consensus. This analysis can also reveal how knowledge is shared within the team, potentially uncovering bottlenecks or knowledge silos that could impact the team's efficiency and effectiveness. The difficulty in analyzing developer conversations lies in the

informal and unstructured nature of these interactions. They often contain a mix of technical jargon, slang, and shorthand, all of which can vary widely between different communities or even teams within the same community. To illustrate the challenges encountered when qualitatively analyzing developer conversations, many discussions on platforms such as Stack Overflow [51] remain unanswered, presenting a challenge in qualitative analysis as it may not be clear whether issues were resolved outside the forum or not, leaving the actual impact of such discussions ambiguous.

Developer Interactions with LLMs The advent of LLMs in software development creates new forms of data through developer interactions with these models [54]. Qualitative analysis of these interactions can provide insight into the common challenges developers face, the kind of support they seek from AI, and the gaps in existing documentation or resources. For example, frequent queries about a specific feature or function can signal the need for better documentation or tutorials. Additionally, the tone and framing of queries can reflect user satisfaction or frustration, providing insights into the AI's utility and potential areas for improvement. Analyzing interactions with LLMs presents unique challenges, primarily due to the evolving nature of both the questions posed by developers and the responses generated by AI. This dynamism requires continuous updates to the analytical framework to ensure relevancy and accuracy in interpreting the data. For example, the DevGPT dataset [54], which was curated to explore how software developers interact with ChatGPT, contains a total of almost 30,000 prompts/answers in almost 5,000 conversations, clearly showing that most conversations take many turns and require thorough analysis to understand the evolving needs of developers and AI responses.

To truly benefit from these qualitative artifacts, it is crucial to approach their analysis systematically. Proper qualitative data analysis can uncover the stories behind the data, providing insights that are not immediately apparent through quantitative methods alone. This requires a detailed understanding of the coding process, an appreciation of the context within which the software is developed, and a keen eye for the subtleties in the data. Successful applications of qualitative analysis in software engineering research illustrate how combining these different artifacts can lead to richer, more holistic insights into software development practices and challenges.

For example, a study by Fan et al., titled "'My GitHub Sponsors Profile is Live!' Investigating the Impact of Twitter/X Mentions on GitHub Sponsors," explores the interplay between social media interactions and financial support mechanisms in open-source software [15]. By qualitatively analyzing tweets linking to GitHub Sponsors profiles, the study highlights how these social media endorsements can significantly impact sponsorship outcomes, providing a deeper understanding of the socioeconomic factors influencing developer support on platforms like GitHub. This integration of data from different sources such as GitHub and Twitter exemplifies the potential of qualitative methods to reveal complex dynamics in software

development. We will refer to examples of the qualitative coding conducted in this study throughout the remainder of this chapter.

3 The Basics of Qualitative Coding

Qualitative coding is a fundamental process in qualitative data analysis, especially in the context of software engineering, where textual data is abundant [47]. It involves categorizing and interpreting textual data to glean insights and patterns. This process is not linear; it is iterative, often requiring researchers to review data and codes multiple times, refine, and interpret as understanding deepens [42]. This section introduces the basics of qualitative coding, discussing various coding methods such as open, axial, and selective coding, each of which plays a crucial role in qualitative analysis [8].

Iterative Process Qualitative coding is a dynamic process that involves moving between data and codes, ensuring that the understanding of the data evolves and deepens with each review [42]. Initially, the codes may be broad and somewhat vague, but as the researcher revisits the data, these codes become more refined and precise. This iterative cycle is essential to ensure that the final set of codes represents the depth and complexity of the data. It allows for the identification of new patterns or themes that may not have been apparent in the initial readings and for the modification or merging of existing codes as the researcher's understanding of the data matures.

As an example, in the study mentioned previously by Fan et al. [15], the first step of qualitatively coding tweets related to open-source sponsorship was unstructured, taking note of anything that appeared to be of interest in the data. In a subsequent phase, the initial question of "what is the nature of these tweets?" was refined into several subquestions, including an investigation of the relationship between the tweet author and users on GitHub, the timing of the tweet in relation to activities on GitHub, the nature of responses to the tweets, and the type of content.

Open Coding Open coding is where the coding process begins and is all about immersion in the data [8]. Researchers approach data with an open mind, allowing the data to speak for themselves. This stage is exploratory and creative, involving a thorough line-by-line analysis of the text. The aim is to label the data with as many codes as necessary to describe all the nuances and facets of the data. These codes are often descriptive and can be phrases or words that encapsulate what the researcher believes to be significant in the text. It is a process of breaking down the data into manageable coded pieces which can then be further analyzed and categorized.

An important consideration in qualitative coding is whether to start with a preexisting taxonomy or framework (deductive approach) or to build the categories from the ground up based on the data (inductive or grounded approach) [50]. Using a preexisting taxonomy can provide a structured starting point, especially useful in fields where established frameworks exist. It can help to organize the data more

systematically from the outset and can guide the analysis toward specific areas of interest. However, it is important to remain open to emerging themes that may not fit neatly into the preestablished categories. On the other hand, starting without a predefined taxonomy allows the data to dictate the categories (grounded theory approach). This approach can be more flexible and open to novel insights but may require more iterative refinement to develop a coherent categorization structure. Both approaches have their merits, and the choice largely depends on the research objectives, the nature of the data, and the theoretical context of the study. It is essential to recognize the influence of the chosen approach on the coding process and to be vigilant about not forcing the data into preconceived categories, thereby potentially overlooking novel or unexpected insights. Moreover, researchers may sometimes find it beneficial to blend both approaches, using a preexisting taxonomy as a starting point while remaining open to emerging themes and categories, thereby harnessing the strengths of both deductive and inductive reasoning.

In the aforementioned study, open coding of the content of tweets related to open-source sponsorship brought up a wide range of codes. For example, a tweet such as "You can now feed my coffee addiction on Github Sponsors. So that I can continue debugging your systemd issues on NixOS" could be coded with anything from a very specific "GitHub user using sponsorship income to buy coffee" to a much wider "support for open-source contributions." This initial broad coding helps identify key themes such as financial support, motivation for contributions, and community engagement within the open-source ecosystem. As the analysis progresses, these broad categories can be further refined and subdivided into more specific themes, providing a detailed map of the various factors that influence open-source sponsorship. The final coding guide included categories such as "generic advertisement," "advertisement with new functionality," and "donation appreciation" to capture a wide variety of content.

Axial Coding Axial coding takes the process a step further by beginning to organize these codes into meaningful categories or themes [8]. It is about making connections between different codes identified during open coding and seeing how they relate to each other. This stage often involves creating visual representations, such as diagrams or flowcharts, to help one see the relationships between codes. Axial coding helps transform a large set of disparate codes into a smaller set of categories or themes, each representing a key aspect of the data. It is about understanding the "axis" around which the codes revolve, hence the term "axial." This stage is crucial for transitioning from descriptive to analytical coding, shifting the focus from what the data are to what they mean.

Selective Coding Selective coding is the final integration and refinement of categories and themes identified during axial coding [8]. Here, the researcher selects one or several "core" categories and systematically relates all other categories to these core ones. This stage is about building a coherent and cohesive narrative or theoretical framework that explains the phenomena represented by the data. It involves determining which categories are central to the narrative and how other categories and codes support, enhance, or relate to this central theme. Selective

coding is about distilling the complexity of data into a clear, understandable, and coherent story or framework. It is the stage in which the researcher begins to understand the larger picture and the deeper meanings within the data.

4 Designing a Coding Guide

Although coding guides are valuable tools in many qualitative research contexts, their necessity and application can vary depending on the specific methodologies and goals of a study. In scenarios where consistency between different analysts is critical—such as in large-scale studies or when preparing for systematic reviews—a well-defined coding guide is indispensable. It helps ensure that all researchers interpret and code the data in a uniform manner, which is essential for the reliability and validity of the findings [32]. In addition, a coding guide serves as a detailed manual that provides clear guidelines and definitions, facilitating a more structured and comprehensible analysis process [37].

However, in more exploratory or interpretative forms of qualitative research, such as phenomenology or certain forms of ethnography, the use of a rigid coding guide might be less appropriate. In these cases, the flexibility of the coding process allows researchers to adapt and evolve their analytical frameworks based on emerging data insights, which is crucial to a deep understanding of complex human experiences and behaviors [24]. Here, a coding guide might instead take the form of a more flexible set of principles or thematic directions rather than strict coding rules.

Regardless of the approach, a coding guide can also be an invaluable training resource. It provides structured guidance and examples for new researchers or team members, ensuring that they are well equipped to contribute effectively to the research. In addition, as a form of documentation, a coding guide records the analytical decisions made during the study, improving the transparency and reproducibility of the research [35]. Whether used as a strict procedural document or as a flexible analytical aid, the role of a coding guide should be tailored to the specific needs and objectives of the qualitative analysis being conducted.

Creating a coding guide starts with a thorough understanding of the data. This understanding is critical to identify potential codes and themes. The codes and categories are then clearly and concisely defined, ensuring that each code captures the essence of what it represents without being ambiguous.

The guide structure comes next, with codes organized into a framework that shows the relationships between themes and categories. But a coding guide is more than its structure; it is also about setting clear rules for code application. These rules help researchers navigate ambiguities and make decisions when data do not fit easily into predefined categories.

Examples are essential to a coding guide. They help coders understand how to apply each code practically, clarifying definitions and rules with real-world references [35]. However, a coding guide is not fixed; it is tested against a sample of data to refine codes, definitions, and rules as needed.

The final step in creating a coding guide is documenting the entire process. This document tells the story of how the guide was developed, highlighting the thought and rigor put into it, which adds to the research's credibility and transparency.

In software development, coding guides can vary widely, each tailored to specific data types. For example, a Code Review Comments Guide categorizes feedback during code reviews, providing insight into discussion quality and dynamics. A Commit Messages Guide can reveal patterns in how changes are communicated. A Bug Report Analysis Guide can identify common user issues and expectations, while a Developer Interaction Guide can analyze team communication and decision-making patterns.

Designing a coding guide requires precision, insight, and foresight. It is essential to ensure that the qualitative analysis is systematic, structured, and insightful. A well-designed coding guide can decode complex data within software repositories, turning intricate narratives into insights that drive software development forward.

The complete coding guide in the example study mentioned above is included in the publication [15] and included entries such as:

- Advertisement with new functionality: "This tweet explicitly advertises the author's own GitHub Sponsors profile while mentioning new functionality of an open-source project."
- Generic advertisement: "This tweet advertises the tweet author's own GitHub Sponsors profile (use this code if the tweet does not fit the other advertisement categories)."
- Sponsor template: "This tweet contains GitHub's template for advertising one's own GitHub Sponsors profile: 'My GitHub Sponsors profile is live! You can sponsor me to support my open source work' with no or minor changes."

These examples showcase the use of concrete examples in a coding guide, providing coding instructions (use this code if...) that help coders apply each code practically, thereby clarifying definitions and rules with real-world references. This approach not only aids in training new researchers or team members but also ensures that the coding process remains transparent and consistent across different datasets and research objectives.

5 Quality in Qualitative Data Analysis

In qualitative research, traditional quality metrics used in quantitative research, such as validity and reliability, assume a different complexion. The inherent subjectivity and depth of understanding sought in qualitative analysis require a different set of criteria to assess quality [31]. Validity, crucial in quantitative research, often focuses on the accuracy of measurements and the effectiveness of statistical inference. However, in qualitative research, where understanding and interpretation are central, the notion of validity does not quite capture the essence of what quality entails. Instead, qualitative researchers focus on the trustworthiness and rigor of their

studies, which is where credibility, transferability, dependability, and confirmability come into play [31].

Credibility Credibility refers to the confidence that can be placed in the truth of research findings [36]. In qualitative research, this is similar to the concept of internal validity in quantitative research but with a focus on the richness, depth, and relevance of data and analysis. Achieving credibility involves prolonged engagement with the data, ensuring that the findings are deeply rooted in the data itself. Techniques such as member verification, where participants review and validate findings, and triangulation, where multiple data sources or methods are used to cross-verify the findings, improve the credibility of the research [36].

Transferability Transferability refers to the extent to which the findings of qualitative research can be applied in other contexts or with other groups [19]. Unlike the quantitative concept of external validity, transferability does not imply a broad generalizability. Instead, it acknowledges the uniqueness of each context and focuses on providing rich and detailed descriptions of the research context and the participants, allowing others to judge the applicability of the findings to their own contexts [19]. A detailed description is a key strategy, providing a comprehensive account of the research setting and assumptions, allowing others to fully understand the context and assess the potential transferability of the findings.

Dependability Dependability is concerned with the consistency of research findings over time and is similar to the notion of reliability in quantitative research [41]. However, given the dynamic and evolving nature of qualitative research, dependability does not imply stability, but rather an understanding of how and why the findings might change. Ensuring dependability involves creating an audit trail, detailed documentation of the research process, decisions, and changes that occur during the study [41]. This allows for an external audit, where an independent examiner reviews the process and the findings, ensuring that the research was conducted rigorously and that the findings are well-substantiated.

Confirmability Confirmability is the qualitative counterpart to objectivity in quantitative research [16]. It refers to the degree to which the findings are shaped by the respondents and not by researcher bias, motivation, or interest. This is achieved through reflexivity, where researchers continuously examine and account for their own biases, preferences, and influence throughout the research process [16]. A reflexive journal, where researchers document their reflections, decisions, and challenges, can be a valuable tool for enhancing confirmability, ensuring that the findings accurately represent the data and not the researcher's preconceptions.

In addition to these criteria, it is sometimes beneficial to quantify the reliability of qualitative analysis by calculating inter-rater agreement. This involves having multiple researchers code the same data independently and then comparing how consistently they apply the codes and categories [31]. A high level of agreement among different raters can reinforce the dependability of the coding process, indicating that the interpretations are not solely dependent on the perspective of a single researcher. However, it is essential to balance this quantitative measure with

the qualitative insights and depth that are central to qualitative research. The goal is not to force a quantitative framework onto qualitative data, but to use inter-rater agreement as one of several tools to enhance the trustworthiness and rigor of the analysis.

6 Practical Approaches and Tools for Analysis

In qualitative data analysis, the tools and methods used to examine and understand the data play an important role. However, it is important to note that the depth and quality of the research depend largely on the systematic and thoughtful approach of the researcher. This section discusses the key features of qualitative analysis tools, highlighting that simpler tools, such as Excel or pen-and-paper methods, can sometimes be most effective.

A good qualitative analysis tool should complement the researcher's work, offering features that align with the project's needs, and enhance the analysis process. Essential features include strong data organization and management. The tool should serve as a reliable data repository, supporting efficient categorization, coding, tagging, and retrieval. Visualization features are another important component, as they can reveal patterns and trends that might not be immediately apparent in the raw data. The ability to represent data in formats such as coding trees, frequency graphs, and relationship networks can provide quick and clear insights.

Collaboration is also a vital part of qualitative analysis in complex fields like software engineering. Tools that promote teamwork, allow shared projects, and support clear communication can significantly improve analysis. Features that enable tracking of changes, such as comment functions and audit trails, can provide valuable information and enhance the collective understanding of the data.

The tool's ability to handle various types of data, from textual bug reports to forum discussions and code comments, is also crucial. Software data often includes mixed text, comprising both natural language and code snippets. It is essential that the tool be versatile and capable of accurately representing this mixed-text format. Tools that cannot effectively manage and display the interplay between natural language and code may not be well suited for analyzing software data. Therefore, the selected tool should be capable of integrating and analyzing different data formats, including mixed text, to offer a comprehensive analysis platform.

However, it is essential to remember that the complexity of a tool does not necessarily equate to a more profound or accurate analysis. Tools should simplify and streamline the process, not complicate it or distract from the primary data. Sometimes, simpler tools can be the most effective. For example, Excel can be highly efficient for organizing, coding, and analyzing data due to its straightforward interface. Traditional methods such as the use of highlighters, sticky notes, and pen and paper can also be highly effective, offering a direct and engaging way to interact with data.

In the end, the true quality of qualitative analysis is not determined by the tools used but by the researcher's thorough and thoughtful approach. The ideal tool or method should integrate smoothly into the analytical process, enabling the researcher to deeply understand and interpret the data. The effectiveness of a tool is measured by its ability to become an integrated part of the researcher's cognitive and analytical processes, allowing a deeper and more nuanced understanding of the data.

Although simpler tools are often highly effective, there are several specialized software tools designed for more complex qualitative data analysis tasks. Here are a few notable examples:

- NVivo (https://lumivero.com/products/nvivo/): This powerful tool for qualitative and mixed methods research helps organize and analyze nonnumerical or unstructured data. It supports deeper data analysis and helps to reveal trends and patterns with features for coding, querying, and linking different datasets.
- ATLAS.ti (https://atlasti.com/): Known for its robust capabilities for qualitative coding and analysis, this tool is suitable for a wide range of data types, including text, multimedia, and geographic or spatial data. It features a user-friendly interface and visual tools to map connections between themes.
- MAXQDA (https://www.maxqda.com/): Ideal for software engineering projects that may involve diverse data sources, MAXQDA supports text, audio, and video data. It offers advanced coding tools, multimedia file management, and visualization options that aid in the structured analysis of qualitative data.
- Dedoose (https://www.dedoose.com/): A web-based application providing features for mixed methods research, Dedoose excels in projects requiring both qualitative and quantitative analysis. Its key strengths include ease of use, affordability, and strong data visualization capabilities.
- HyperRESEARCH (https://www.researchware.com/products.html): Known for its flexibility, this tool supports a variety of text and media data formats and offers a straightforward approach to coding and retrieving data. It also allows for the building of complex filters and reports.

The integration of large language models (LLMs) into software engineering research presents significant opportunities and challenges [3]. LLMs, such as GPT-4 and ChatGPT, are becoming established in various disciplines, including qualitative software engineering research, where they offer the potential to enhance the research process. These models can optimize certain aspects of research, such as data coding and analysis, helping researchers handle large volumes of data more efficiently.

However, the adoption of LLMs in qualitative research also brings concerns about their ability to fully replace human roles, a notion debunked by empirical findings. The future of qualitative software engineering research is envisioned as a collaborative one, where LLMs and human researchers work together to advance the field. While LLMs show promise in aiding the qualitative research process, the critical and irreplaceable role of human researchers remains central. Humans are essential for ensuring ethical conduct, maintaining the validity and reliability of research findings, and managing the appropriate dissemination of results.

Furthermore, while LLMs can enhance the efficiency of data analysis, their limitations in capturing the nuanced understanding inherent to human researchers are notable. The "human touch" remains vital in interpreting and understanding qualitative data, a sentiment echoed in all anthropological and sociological disciplines. Furthermore, ethical considerations surrounding the use of LLMs, including issues of data privacy and intellectual property rights, require rigorous scrutiny to ensure that their integration into research practices adheres to the highest standards of ethical conduct.

7 Common Pitfalls and Best Practices

Educators who teach qualitative data analysis in software engineering should be particularly aware of potential pitfalls and adhere to best practices. This structured approach ensures that the analysis taught is not only insightful but also methodologically sound [33]. In the following, we outline common pitfalls along with the corresponding best practices, emphasizing the need for a balanced and rigorous approach in the educational context.

Confirmation Biases One of the most common pitfalls is confirmation bias, where researchers may subconsciously favor data that align with their initial hypotheses, neglecting data that contradict them [34]. This bias can significantly skew the analysis. Educators should stress the importance of approaching data with an open mind, actively seeking disconfirming evidence, and engaging multiple researchers to provide diverse perspectives. This collective approach can offer a balanced interpretation that minimizes the influence of individual biases.

Data Overload Researchers often face the challenge of data overload, where the vast volume of data can lead to superficial analysis or overlooked nuances [40]. For educators, it is crucial to teach effective data management strategies. Organizing data into manageable segments, employing a robust coding framework, and focusing on depth rather than breadth in analysis can prevent being overwhelmed. Furthermore, selective and purposeful data sampling, focusing on data that are most relevant to the research questions, ensures clarity and focus in the analysis.

Subjectivity The subjective nature of qualitative analysis, while a strength, can also be a pitfall if it leads to overly subjective interpretations. Educators must guide students to improve objectivity and credibility by establishing clear and transparent processes. Documenting the decision-making process, providing well-defined codes with examples, and maintaining a reflexive journal to note assumptions and biases can improve the objectivity and traceability of the analysis.

Inconsistency in Coding Maintaining consistency in coding, particularly in collaborative projects, can be challenging. Inconsistencies in coding can cause confusion and compromise the integrity of the analysis [28]. Best practice involves holding regular team discussions to align on the coding framework, employing

intercoder reliability measures, and creating a detailed and comprehensive coding guide. These steps ensure that the coding remains consistent and reliable, regardless of the researcher.

Overinterpretation of Data Researchers might fall into the trap of overinterpreting the data, reading more into it than is actually there. This can lead to conclusions that are not strongly supported by the data. Educators should emphasize the maintenance of a balance between creativity and discipline. Ensure that interpretations are directly linked to the data. Peer debriefing, where researchers discuss their interpretations with colleagues, can provide a check against overinterpretation, ensuring that conclusions are well-grounded in the data.

Ignoring Negative Cases Researchers can ignore or underestimate data that do not fit emerging patterns or themes, known as negative or deviant cases [18]. This can lead to a one-sided understanding of the data. Educators should emphasize the importance of actively looking for and considering negative cases. These instances can provide valuable information and help refine and challenge emerging theories or patterns. Incorporating negative cases strengthens the analysis by providing a more comprehensive understanding of the data.

Lack of Transparency Without a clear exposition of the methods and processes used in data collection and analysis, research can lack transparency, making it difficult for others to understand, evaluate, or replicate the study [31]. Educators must emphasize the necessity of maintaining an audit trail. An audit trail includes detailed records of the data collection methods, the analysis process, the decision-making points, and any changes made throughout the investigation. It should also document the rationale behind methodological choices, coding decisions, and the evolution of the researcher's understanding. By preserving this level of detail, an audit trail not only enhances the transparency of the investigation but also substantiates its credibility, allowing other researchers to follow the progression of the study and understand the context and reasoning behind the findings.

Overemphasis on Tools Rather Than Concepts Educators can fall into the trap of focusing too heavily on teaching specific qualitative analysis tools, neglecting the foundational concepts that underpin qualitative analysis. This can lead to students who are proficient in using tools but may lack the ability to conduct analysis independently of specific software. Educators should balance tool instruction with robust discussions on theoretical frameworks and the principles of qualitative research to ensure that students understand the "why" behind the "how."

Lack of Real-World Examples It is vital for educators to incorporate real-world examples to bridge theory with practice. Using simplified or hypothetical cases can fail to convey the complexity and nuances of actual qualitative analysis. Incorporating case studies from recent research, including pitfalls and successes, can provide students with a more realistic perspective on the challenges and dynamics of qualitative data analysis.

Neglecting the Ethical Dimensions Qualitative data often involve sensitive information, and ethical considerations are paramount. Educators may sometimes overlook the importance of discussing the ethical implications involved in data collection, analysis, and reporting. It is essential for educators to integrate ethics into the curriculum, ensuring that students understand how to ethically handle data, respect participant confidentiality, and report findings responsibly.

Getting Overwhelmed by Rich Data Qualitative data can be rich and complex, and researchers may feel overwhelmed by depth and volume, which could lead to burnout or analysis paralysis. Best practice is for educators to teach students to take a phased approach to the analysis, breaking the process into manageable stages. Regular breaks and reflections can help to maintain focus and perspective. Using visualizations and tools can also help to manage and organize data effectively.

In essence, while qualitative analysis presents unique challenges, being aware of these pitfalls and adhering to best practices can lead to insightful, credible, and valuable analysis. It is about striking the right balance between the richness of subjective insights and the rigor of systematic, methodical analysis, ensuring that the conclusions drawn from qualitative data are not only profound and nuanced but also methodologically robust and reliable.

8 Lessons Learned About Teaching Qualitative Data Analysis

Teaching qualitative data analysis, particularly in the dynamic field of software engineering, offers a unique set of challenges and opportunities. A fundamental lesson is the importance of contextual learning. Students often understand the concepts of qualitative analysis more deeply when these concepts are framed within real-world scenarios and applications. Therefore, it is recommended to integrate case studies, use actual data from software projects, and refer to recent research within the curriculum. This method not only makes the learning experience more engaging but also clearly demonstrates the practical relevance and application of qualitative analysis skills in real-world settings.

Encouraging critical thinking is crucial in qualitative analysis, a field characterized by its interpretative and subjective nature. Students should be nurtured to develop the ability to critically review data and to deeply reflect on their analytical choices and processes. As a best practice, educators should design courses that force students to justify their coding decisions, interpret data with care, and contemplate alternative perspectives. Students should be asked to question their preconceptions and recognize how their biases might color their analysis.

Understanding the iterative nature of qualitative data analysis is another key lesson. Unlike linear progression often portrayed in textbooks, qualitative analysis is a recursive process, involving continuous cycles of coding, categorization, and theme development. To impart this lesson effectively, educators should provide students

with practical opportunities to participate in this iterative process. Assignments that require revisiting and refining analyses, supplemented with stages of constructive feedback, can demystify the iterative nature of qualitative analysis and cultivate students' skills in this area.

Although the intellectual process of analysis is crucial, familiarity with various qualitative data analysis tools can substantially increase the students' analytical efficiency and depth. Educators are encouraged to introduce students to a spectrum of tools, from basic software such as Excel to more sophisticated qualitative analysis programs. However, it is important that the focus on tools does not overshadow the foundational principles and practices of qualitative analysis.

The ability to collaborate effectively and communicate the findings clearly is important in qualitative analysis, which often involves teamwork. To foster these essential skills, educators are recommended to incorporate group projects and peer review sessions into their courses. Emphasis should be placed on the importance of transparency in the research process and the ability to articulate one's analytical approach and insights clearly and coherently.

Lastly, understanding and adhering to ethical standards is a critical component of qualitative research, particularly when human subjects are involved. Educators must ensure that students not only grasp the ethical dimensions of their research but also know how to practically apply these principles. This includes managing confidentiality, navigating the informed consent process, and handling data responsibly. Integrating research ethics modules into the curriculum can be beneficial. These modules could include real-world examples and case studies that illustrate ethical dilemmas that researchers often face. In addition, offering practical exercises, such as drafting consent forms or developing a data management plan, can provide students with hands-on experience in addressing these ethical issues.

9 Quality Checklist

Summarizing the principles of qualitative data analysis in software engineering, we recognize the intricacies and depth involved in this analytical process. It requires a systematic and thoughtful approach to ensure that the analysis is thorough and methodologically sound. To support this process, we introduce the quality checklist shown in Table 1. This checklist consists of key questions designed to guide students, researchers, and practitioners in maintaining rigor and attention to detail throughout each phase of their qualitative analysis.

10 Conclusion

Qualitative data analysis in software engineering entails a thorough examination and interpretation of data, transforming these insights into actionable information

Table 1 Qualitative analysis checklist

Stage	Checklist item
Data familiarization	Have you thoroughly immersed yourself in the data?
	Have you continuously reviewed the data to deepen your understanding?
Open coding	Have you started coding without preconceived notions, allowing the data to inform your codes?
	Are your codes descriptive and deeply rooted in the data?
Axial coding	Have you examined the relationships between codes and organized them into coherent categories?
	Are you developing a narrative or framework that connects these categories meaningfully?
Selective coding	Have you pinpointed core categories and related all other categories to these?
	Is there a clear, overarching narrative or theory emerging from your analysis?
Iterative refinement	Are you revisiting your codes and categories and refining and redefining them as understanding matures?
	Is the iterative process contributing to a deeper, more nuanced understanding of the data?
Collaboration and peer review	Have you engaged with peers or mentors to discuss your codes and findings?
	Are you open and receptive to feedback, willing to reconsider aspects of your analysis based on this input?
Reflection and reflexivity	Have you maintained a record of your analytical decisions, reflections, and challenges faced?
	Are you critically examining your own biases and their influence on your interpretation?
Ethical consideration	Have you ensured the confidentiality and ethical treatment of your data?
	Are you adhering to ethical standards and guidelines in your analysis and reporting?
Transparency in documentation	Have you kept a clear and detailed record of your methodology and findings?
	Is your documentation complete enough to allow others to understand, evaluate, or replicate your study?
Communication findings	Are you presenting your findings clearly and logically, articulating the developed narrative?
	Have you provided ample evidence, like direct quotes or clear examples, to support your findings?

that can refine the software development process. This chapter has explored the rich variety of qualitative data present in software repositories, such as source code comments and developer interactions, each offering unique insights into the complexities of software projects.

Throughout this discussion, we have explored qualitative coding processes, including open, axial, and selective coding, and emphasized the critical importance of revisiting and reflecting on the data to deepen understanding and refine analysis. The iterative nature of this analysis is essential for developing a comprehensive understanding of the underlying patterns and themes.

A well-crafted coding guide has been highlighted as more than just a tool: It serves as a crucial instructional resource that helps researchers, particularly educators, navigate the complexities of qualitative data. This guide not only supports consistent and accurate coding practices but also serves as an invaluable training tool for new researchers and students. By outlining clear guidelines and examples, it ensures that the analysis remains structured and transparent.

We have underscored the foundational qualities of qualitative research: credibility, transferability, dependability, and confirmability, stressing the importance of conducting studies that are both trustworthy and thorough. The discussion of practical tools and methods has noted that sometimes simpler approaches, such as using Excel or traditional pen-and-paper techniques, can be highly effective, especially in educational settings where resources may be limited.

Furthermore, this chapter has addressed common pitfalls and best practices in qualitative analysis, advocating for a structured and balanced approach that can yield insightful and reliable results. Critical insights on teaching qualitative data analysis have been shared, highlighting the need to provide real-world context, fostering critical thinking, and emphasizing the iterative nature of analysis. These elements are crucial in equipping students not only with technical skills but also with a deeper understanding of the research process.

In conclusion, qualitative data analysis in software engineering is not merely data interpretation; it is about crafting a narrative that is methodologically sound, contextually rich, and deeply insightful. For educators, this field offers dynamic opportunities to engage students in a reflective and structured inquiry into the nuances of software development, propelled by insightful qualitative analysis. This chapter aims to serve as a foundational resource for educators striving to instill these critical skills in the next generation of software engineers, thereby enhancing both their understanding and their impact on the field.

11 Further Readings

- Steve Adolph, Wendy Hall, and Philippe Kruchten. “Using grounded theory to study the experience of software development”. In: *Empirical Software Engineering* 16 (2011), pp. 487–513 [1]

- Sebastian Baltes and Paul Ralph. “Sampling in software engineering research: A critical review and guidelines”. In: *Empirical Software Engineering* 27.4 (2022), p. 94 [2]
- Alan Bryman. *Social research methods*. Oxford university press, 2016 [6]
- Joanna F DeFranco and Phillip A Laplante. “A content analysis process for qualitative software engineering research”. In: *Innovations in Systems and Software Engineering* 13 (2017), pp. 129–141 [10]
- Norman K Denzin and Yvonna S Lincoln. *The Sage handbook of qualitative research*. sage, 2011[11]
- Yvonne Dittrich, Michael John, Janice Singer, and Bjørnar Tessem. “Editorial for the special issue on qualitative software engineering research”. In: *Information and software technology* 49.6 (2007), pp. 531–539 [12]
- Tore Dyba, Barbara A Kitchenham, and Magne Jorgensen. “Evidence-based software engineering for practitioners”. In: *IEEE software* 22.1 (2005), pp. 58–65 [13]
- Steve Easterbrook, Janice Singer, Margaret-Anne Storey, and Daniela Damian. “Selecting empirical methods for software engineering research”. In: *Guide to advanced empirical software engineering* (2008), pp. 285–311 [14]
- Uwe Flick. “An introduction to qualitative research”. In: (2022) [17]
- Barney Glaser and Anselm Strauss. *Discovery of grounded theory: Strategies for qualitative research*. Routledge, 2017 [20]
- Martyn Hammersley and Paul Atkinson. *Ethnography: Principles in practice*. Routledge, 2019 [21]
- Rashina Hoda, James Noble, and Stuart Marshall. “Using grounded theory to study the human aspects of software engineering”. In: *Human Aspects of Software Engineering*. 2010, pp. 1–2 [25]
- AJ Ko. “Understanding software engineering through qualitative methods”. In: *Making Software* (2010), pp. 55–63 [27]
- Per Lenberg, Robert Feldt, Lucas Gren, Lars Göran Wallgren Tengberg, Inga Tidefors, and Daniel Graziotin. “Qualitative software engineering research: Reflections and guidelines”. In: *Journal of Software: Evolution and Process* (2023), e2607 [29]
- Per Lenberg, Robert Feldt, Lars Göran Wallgren Tengberg, Inga Tidefors, and Daniel Graziotin. “Behavioral software engineering-guidelines for qualitative studies”. In: *arXiv preprint arXiv:1712.08341* (2017) [30]
- Paul Ralph. “Toward methodological guidelines for process theories and taxonomies in software engineering”. In: *IEEE Transactions on Software Engineering* 45.7 (2018), pp. 712–735 [38]
- Paul Ralph, Nauman bin Ali, Sebastian Baltes, Domenico Bianculli, Jessica Diaz, Yvonne Dittrich, Neil Ernst, Michael Felderer, Robert Feldt, Antonio Filieri, et al. “Empirical standards for software engineering research”. In: *arXiv preprint arXiv:2010.03525* (2020) [39]
- Mary Shaw. “What makes good research in software engineering?” In: *International Journal on Software Tools for Technology Transfer* 4 (2002), pp. 1–7 [44]

- Dag IK Sjoberg, Tore Dyba, and Magne Jorgensen. “The future of empirical methods in software engineering research”. In: *Future of Software Engineering (FOSE’07)*. IEEE. 2007, pp. 358–378 [45]
- Klaas-Jan Stol and Brian Fitzgerald. “The ABC of software engineering research”. In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 27.3 (2018), pp. 1–51 [46]
- Anselm Strauss and Juliet Corbin. “Basics of qualitative research techniques”. In: (1998) [49]
- Claes Wohlin and Aybüke Aurum. “Towards a decision-making structure for selecting a research design in empirical software engineering”. In: *Empirical Software Engineering* 20 (2015), pp. 1427–1455 [52]
- Claes Wohlin, Martin Höst, and Kennet Henningsson. “Empirical research methods in web and software engineering”. In: *Web engineering* (2006), pp. 409–430 [53]

References

1. Adolph, S., Hall, W., Kruchten, P.: Using grounded theory to study the experience of software development. Empir. Softw. Eng. **16**, 487–513 (2011)
2. Baltes, S., Ralph, P.: Sampling in software engineering research: a critical review and guidelines. Empir. Softw. Eng. **27**(4), 94 (2022)
3. Bano, M., Hoda, R., Zowghi, D., Treude, C.: Large language models for qualitative research in software engineering: exploring opportunities and challenges. Autom. Softw. Eng. **31**(1), 8 (2024)
4. Bettenburg, N., Just, S., Schröter, A., Weiss, C., Premraj, R., Zimmermann, T.: What makes a good bug report? In: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 308–318 (2008)
5. Bird, C., Nagappan, N., Gall, H., Murphy, B., Devanbu, P.: Putting it all together: using socio-technical networks to predict failures. In: 2009 20th International Symposium on Software Reliability Engineering. IEEE, Piscataway (2009), pp. 109–119
6. Bryman, A: Social Research Methods. Oxford University Press, Oxford (2016)
7. Buse, R.P., Weimer, W.R.: Automatically documenting program changes. In: Proceedings of the 25th IEEE/ACM International Conference on Automated Software Engineering, pp. 33–42 (2010)
8. Charmaz, K: Constructing Grounded Theory. Sage, Beverly Hills (2014)
9. Creswell, J.W., Poth, C.N.: Qualitative inquiry and research design: choosing among five approaches. Sage Publications (2016)
10. DeFranco, J.F., Laplante, P.A.: A content analysis process for qualitative software engineering research. Innov. Syst. Softw. Eng. **13**, 129–141 (2017)
11. Denzin, N.K., Lincoln, Y.S.: The Sage Handbook of Qualitative Research. Sage, Beverly Hills (2011).
12. Dittrich, Y., John, M., Singer, J., Tessem, B.: Editorial for the special issue on qualitative software engineering research. Inform. Softw. Technol. **49**(6), 531–539 (2007)
13. Dyba, T., Kitchenham, B.A., Jorgensen, M.: Evidence-based software engineering for practitioners. IEEE Softw. **22**(1), 58–65 (2005)
14. Easterbrook, S., Singer, J., Storey, M.A., Damian, D.: Selecting empirical methods for software engineering research. In: Guide to Advanced Empirical Software Engineering, pp. 285–311 (2008)

15. Fan, Y., Xiao, T., Hata, H., Treude, C., Matsumoto, K.: My GitHub Sponsors profile is live! Investigating the impact of Twitter/X mentions on GitHub sponsors. In: Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, pp. 1–12 2024
16. Finlay, L.: "Outing" the researcher: the provenance, process, and practice of reflexivity. Qualitat. Health Res. **12**(4), 531–545 (2002)
17. Flick, U.: An introduction to qualitative research (2022)
18. Flyvbjerg, B.: Five misunderstandings about case-study research. Qualitat. Inquiry **12**(2), 219–245 (2006)
19. Geertz, C.: The Interpretation of Cultures. Basic books (1973)
20. Glaser, B., Strauss, A.: Discovery of Grounded Theory: Strategies for Qualitative Research. Routledge, London (2017)
21. Hammersley, M, Atkinson, P.: Ethnography: Principles in Practice. Routledge, London (2019)
22. Hassan, A.E.: The road ahead for mining software repositories. In: 2008 Frontiers of Software Maintenance, pp. 48–57. IEEE, Piscataway (2008)
23. Hata, H., Novielli, N., Baltes, S., Kula, R.G., Treude, C.: GitHub Discussions: an exploratory study of early adoption. Empir. Softw. Eng. **27**, 1–32 (2022)
24. Hoda, R.: Socio-technical grounded theory for software engineering. IEEE Trans. Softw. Eng. **48**(10), 3808–3832 (2021)
25. Hoda, R., Noble, J., Marshall, S.: Using grounded theory to study the human aspects of software engineering. In: Human Aspects of Software Engineering, pp. 1–2 (2010)
26. Kagdi, H., Collard, M.L., Maletic, J.I.: A survey and taxonomy of approaches for mining software repositories in the context of software evolution. J. Softw. Maintenance Evol. Res. Practice **19**(2), 77–131 (2007)
27. Ko, A.J.: Understanding software engineering through qualitative methods. In: Making Software, pp. 55–63 (2010)
28. Krippendorff, K.: Content Analysis: An Introduction to Its Methodology. Sage Publications (2018)
29. Lenberg, P., Feldt, R., Gren, L., Tengberg, L.G.W, Tidefors, I., Graziotin, D.: Qualitative software engineering research: reflections and guidelines. J. Softw. Evol. Process **36**(6), e2607 (2024)
30. Lenberg, P., Feldt, R., Tengberg, L.G.W., Tidefors, I., Graziotin, D.: Behavioral software engineering-guidelines for qualitative studies. In: arXiv preprint arXiv:1712.08341 (2017)
31. Lincoln, Y.S., Guba, E.G.: Naturalistic Inquiry. Sage (1985)
32. MacQueen, K.M., McLellan, E., Kay, K., Milstein, B.: Codebook development for team-based qualitative analysis. Cam J. **10**(2), 31–36 (1998)
33. Maxwell, J.A.: Qualitative Research Design: An Interactive Approach. Sage Publications (2012)
34. Mays, N.: Quality in qualitative health research. In: Qualitative Research in Health Care, pp. 89–101 (2000)
35. Miles, M.B., Huberman, A.M., Saldana, J.: Qualitative Data Analysis: A Methods Sourcebook. Sage Publications (2013)
36. Morse, J.M., Barrett, M., Mayan, M., Olson, K., Spiers, J.: Verification strategies for establishing reliability and validity in qualitative research. Int. J. Qualitat. Methods **1**(2), 13–22 (2002)
37. Rabiee, F.: Focus-group interview and data analysis. Proc. Nutr. Soc. **63**(4), 655–660 (2004)
38. Ralph, P.: Toward methodological guidelines for process theories and taxonomies in software engineering. IEEE Trans. Softw. Eng. **45**(7), 712–735 (2018)
39. Ralph, P., Ali, N.B., Baltes, S., Bianculli, D., Diaz, J., Dittrich, Y., Ernst, N., Felderer, M., Feldt, R., Filieri, A., et al.: Empirical standards for software engineering research. arXiv preprint arXiv:2010.03525 (2020)
40. Richards, L.: Handling Qualitative Data: A Practical Guide. Sage (2020)
41. Rodwell, M.K.: Social Work, Constructivist Research. Taylor & Francis, London (1998)
42. Saldana, J.M.: The Coding Manual for Qualitative Researchers, 3rd edn. Sage Publications, London (2015)

43. Seaman, C.B.: Qualitative methods in empirical studies of software engineering. IEEE Trans. Softw. Eng. **25**(4), 557–572 (1999)
44. Shaw, M.: What makes good research in software engineering? Int. J. Softw. Tools Technol. Transf. **4**, 1–7 (2002)
45. Sjoberg, D.I., Dyba, T., Jorgensen, M.: The future of empirical methods in software engineering research. In: Future of Software Engineering (FOSE'07), pp. 358–378. IEEE, Piscataway (2007)
46. Stol, K.-J., Fitzgerald, B.: The ABC of software engineering research. ACM Trans. Softw. Eng. Methodol. **27**(3), 1–51 (2018)
47. Stol, K.-J., Ralph, P., Fitzgerald, B.: Grounded theory in software engineering research: a critical review and guidelines. In: Proceedings of the 38th International Conference on Software Engineering, pp. 120–131 (2016)
48. Storey, M.-A., Zagalsky, A., Figueira Filho, F., Singer, L., German, D.M.: How social and communication channels shape and challenge a participatory culture in software development. IEEE Trans. Softw. Eng. **43**(2), 185–204 (2016)
49. Strauss, A., Corbin, J.: Basics of Qualitative Research Techniques (1998)
50. Thomas, D.R.: A general inductive approach for analyzing qualitative evaluation data. Am. J. Eval. **27**(2), 237–246 (2006)
51. Treude, C., Barzilay, O., Storey, M.A.: How do programmers ask and answer questions on the web?(nier track). In: Proceedings of the 33rd International Conference on Software Engineering, pp. 804–807 (2011)
52. Wohlin, C., Aurum, A.: Towards a decision-making structure for selecting a research design in empirical software engineering. Empir. Softw. Eng. **20**, 1427–1455 (2015)
53. Wohlin, C., Höst, M., Henningsson, K.: Empirical research methods in web and software engineering. In: Web Engineering, pp. 409–430 (2006)
54. Xiao, T., Treude, C., Hata, H., Matsumoto, K.: DevGPT: studying developer-ChatGPT conversations. In: Proceedings of the International Conference on Mining Software Repositories (MSR 2024) (2024)

Part III
Experience Reports and Reflections

Teaching Empirical Methods at Eindhoven University of Technology

Alexander Serebrenik and Nathan Cassee

Abstract In this chapter, we share an experience report of teaching a master course on empirical research methods at Eindhoven University of Technology in the Netherlands. The course is taught for ten weeks to a mix of students from different study programs and combines both practical assignments with a closed-book exam. We discuss the challenges of teaching a course on research methods and explain how we address these challenges in the course design. Additionally, we share our lessons learned and the challenges we encountered over several iterations of teaching the course.

1 Introduction

Empirical methods in software engineering are taught in numerous institutions worldwide, from Canada[1] and the United States[2] to Germany,[3] Sweden, and[4]

[1] For example, at University of Victoria https://emse-uvic.github.io/index.html or at University of Toronto https://www.eecg.utoronto.ca/~shuruiz/teaching/ECE1785-2021W/.

[2] For example, at Carnegie Mellon University https://bvasiles.github.io/empirical-methods/ or at North Dakota State University https://catalog.ndsu.edu/search/?P=CSCI%20848.

[3] For example, at Freie Universität Berlin https://www.mi.fu-berlin.de/w/SE/VorlesungEmpirie2022 or Universität des Saarlandes https://cms.sic.saarland/empse_ws_2223/.

[4] For example, at Gothenburg University https://www.gu.se/en/study-gothenburg/empirical-software-engineering-dit246.

A. Serebrenik (✉) · N. Cassee
Eindhoven University of Technology, Eindhoven, The Netherlands
e-mail: a.serebrenik@tue.nl; n.w.cassee@tue.nl

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2024

D. Mendez et al. (eds.), *Handbook on Teaching Empirical Software Engineering*,
https://doi.org/10.1007/978-3-031-71769-7_7

Norway[5] and from Brazil[6] and Uruguay[7] to Australia.[8] While bearing similar names, these courses are different: they target different student populations, engage with larger or smaller groups, and cover different selections of topics.

This experience report reflects on three years of teaching empirical software engineering in the master's program at Eindhoven University of Technology (TU/e), The Netherlands. To accompany this chapter, we also share all teaching materials we have designed for the course. We hope that this report helps colleagues who are teaching similar courses elsewhere.

2 Background: Master Courses at Eindhoven University of Technology

2.1 Student Demographics

Eindhoven University of Technology offers 24 different master programs including computer science and engineering, data science and artificial intelligence, and human-technology interaction. In addition to a limited number of mandatory courses determined by each program, students are free to take any course of their liking. Most of the courses offered, including *empirical methods in software engineering*, are indeed open for students of any master's program; the aforementioned programs, however, explicitly list *empirical methods in software engineering* as one of the recommended courses or even prescribe it to subgroups of their students. Most students registered for the 2023/2024 edition of *empirical methods in software engineering* follow either the computer science and engineering master program (40) or data science and artificial intelligence (21). The remaining students come from such master programs as human-technology interaction (two students), embedded systems, industrial and applied mathematics, artificial intelligence engineering systems, and pre-master program computer science and engineering (one student each). Furthermore, the number of students increases each year: from 41 in 2020 to 51 in 2021, 60 in 2022, and 67 in 2023. Hence, when designing the course, we had to take into account *a relatively large number of students* taking the course.

Moreover, diversity of master programs implies *diversity of the students' educational backgrounds*: e.g., while data science students had limited exposure to

[5] For example, at NTNU https://www.ntnu.edu/studies/courses/IT3010#tab=omEmnet.

[6] For example, at Universidade Tecnológica Federal do Paraná https://www.utfpr.edu.br/cursos/coordenacoes/stricto-sensu/ppgcc-cm/area-academica/disciplinas/engenharia-de-software-empirica or at Universidade Federal do Acre https://portal.ufac.br/ementario/disciplina.action;jsessionid=ED0C1B1B9875484A460ECF2EC64D358D?d=22091.

[7] For example, at Universidad ORT https://fi.ort.edu.uy/92839/33/.

[8] For example, at the University of Adelaidehttps://www.adelaide.edu.au/course-outlines/108290/1/sem-1/.

basic software engineering practices, they have ample experience with training machine learning classifiers and quantitative data analysis. The situation is reversed for computer science students. Students in the human-technology interaction master program often have a bachelor's degree in psychology and, hence, are experienced with conducting quantitative surveys and designing interventions but not with analyzing data from online platforms. Finally, diversity in educational background does not only refer to the academic discipline but also to the previous education: while some students hold a bachelor degree from more scientifically oriented institutions such as Eindhoven University of Technology itself, others hold a bachelor degree from a Dutch university of applied sciences, which is much more focused on the engineering practice. Such students are often much better software developers than their peers trained at scientifically oriented institutions but are less familiar with more abstract, more conceptual ways of thinking.

2.2 Schedule

All master programs are designed for two academic years: students are expected to start their education in September and graduate in August of the following academic year. In practice, however, students sometimes take longer than two years to complete their degrees for various reasons.

Every course is offered during one teaching period, consisting of 7 regular teaching weeks, 1 week that can be used to compensate for the teacher's absence, and 2 exam weeks. During a regular week, each course is allocated two sessions, one in the morning and another one in the afternoon. Each session consists of two academic hours (45 minutes) separated by a 15-minute break. Every course is considered to be 5 ECTS' worth. ECTS, European Credit Transfer and Accumulation System, expresses the "volume of learning based on the defined learning outcomes and their associated workload" [16]. A full-time study during one academic year corresponds to 60 ECTS. According to the Dutch norm, 1 ECTS corresponds to 28 hours of study, i.e., an average student is expected to spend 140 hours studying for a single course. Since the number of contact hours is limited to 28 (7 weeks, two meetings per week, 2 hours per meeting), *the lion's share of the course work should be organized as a self-study*.

Since *empirical methods in software engineering* is an elective subject for most students, they can take it either during their first year or during their second year. However, in practice *empirical methods in software engineering* is usually taken at the beginning of the master's program, when students are not yet working on any research projects. At the same time, it is supposed to prepare the students for conducting a research project leading toward their master's thesis. This imposes the challenge of teaching the students how to conduct empirical research without giving them the feeling that they are practicing a skill that they do not need.

2.3 Assessment

Teachers of courses at Eindhoven University of Technology are free to choose between a written exam, an oral exam, an assessment based on homework assignments, or any combination of those. In case the assessment is based on group projects, part of the grade should be determined based on individual contributions. A written exam can take up to three hours, and the resit is organized during the evening hours after the subsequent teaching period, i.e., ca. 10 weeks after the first attempt. Under special circumstances, upon student's request, a faculty-level committee can ask the teacher to organize another resit; however, such resits are uncommon. Most teachers opt for a combination of an assessment based on homework assignments, often organized as group assignments, and a written exam. Also, here, diversity of educational backgrounds plays a role. Students holding a bachelor's degree from the Netherlands can be expected to be familiar with group assignments: e.g., most of the courses in the computer science bachelor of Eindhoven University of Technology have some form of group projects. However, students holding a bachelor's degree from abroad have not necessarily had the same exposure to group projects.

At the end of the course, students receive a course grade that is an integer between 1 and 10 with 6 being the minimal passing grade. Usually, the final course grade is determined as the weighted average of the grades for individual course components such as the exam and the homework assignments, rounded to the nearest integer. Very high course grades exceeding 9 are uncommon. Grades for the individual course components are between 1 and 10 and can have one digit after the decimal point, e.g., 7.4.

2.4 Resources

The teaching staff of a master course usually consists of one or two teachers, rarely supported by student assistants. For *empirical methods in software engineering*, the only teachers involved are the authors of this chapter.

Finally, students and teachers are encouraged to use Canvas LMS, a learning management system, and ANS-Delft, an assessment platform designed to support paper, digital, and hybrid exams.

2.5 Summary

In summary, the main educational challenge addressed in this course design is **how to teach, in 10 weeks, a self-study-driven, realistic course on empirical software engineering research methods to a group of 50–70 students with no previous**

research experience and diverse backgrounds while considering the limited resources available.

3 *Empirical Methods in Software Engineering*

In this section, we discuss the learning objectives of the course, the teaching activities, and the assessment.

3.1 Learning Objectives

As the first step toward designing the course, we have formulated the following learning objectives:

LO1 Students will be able to formulate and motivate research questions pertaining to software engineering and identify research questions that can and cannot be answered by means of empirical research.
LO2 Students will be able to execute a valid and trustworthy empirical study in software engineering given a state-of-the-art dataset.
LO3 Students will be able to evaluate empirical studies in software engineering using instruments accepted in the field and identify threats to validity and problems with trustworthiness.
LO4 Students will be able to describe the results of empirical studies to professionals not familiar with academic research.
LO5 Students will be able to comprehend the research methods used for empirical studies in software engineering.

We strongly believe that these learning objectives capture the most important aspects of conducting empirical research in software engineering. **LO1**, **LO2**, and **LO3** address the steps inherent in any empirical research and, subsequently, the skills the students will need when conducting empirical research either as part of their study or as part of their further career. However, before students can conduct empirical studies, they must understand the research methods commonly used in empirical studies (**LO5**). Finally, **LO4** relates to the societal obligation of researchers to convey the research findings to practitioners.

3.2 Course Organization

To achieve the learning objectives above, with the aforementioned educational challenges in mind, we have offered an overview course showcasing several different empirical methods and their applications to software engineering research.

The teaching activities of *empirical methods in software engineering* consist of a mix of lectures, workshops, and coffee hours. As customary, *lectures* are used to teach students the theoretical foundations of empirical research methods (**LO5**). In contrast, the *workshops* are more interactive and are intended for students to work on bridging the gap between a more abstract presentation of the research methods and their application in the practice of software engineering research (**LO1**–**LO4**). The *coffee hours* are scheduled during the regular lecture hours and are a platform for students to ask questions about the course and the assignments. Our decision to allocate time to workshops and coffee hours implies that we had to limit the number of topics that could be covered during the course. When selecting the topics for the lectures, we decided to include topics that are orthogonal to the specific choice of a research method (e.g., stating research questions and choosing sampling techniques), as well as topics showcasing the diversity of empirical software engineering research methods (e.g., interviews vs. repository mining, quantitative analysis vs. qualitative analysis). The overview of the course schedule for the 2022 edition can be found in Table 1.

Table 1 Course schedule for the 2022 edition of *empirical methods in software engineering*

When	What	Topic
Nov 15TH 13:30	Lecture	Course introduction: Empiricism and Rationalism
Nov 17TH 08:45	Lecture	Research Questions and Research Strategies
Nov 22ND 13:30	Guest Lecture	Empirical Methods in the Industry
Nov 24TH 08:45	Lecture	Sampling
Nov 25TH 23:59	Deadline	1ST deadline for Design a Study
Nov 29TH 13:30	Workshop	How to Read an Empirical Paper
Dec 1ST 08:45	Lecture	Interviews & Surveys
Dec 6TH 13:30	Lecture	Mining Software Repositories I
Dec 8TH 08:45	Workshop	Mining Software Repositories
Dec 9TH 23:59	Deadline	2ND deadline for Design a Study
Dec 13TH 13:30	Lecture	Quantitative Analysis
Dec 15TH 08:45	Lecture	Qualitative Analysis
Dec 16TH 23:59	Deadline	1ST deadline for Describe a Study
Dec 20TH 13:30	Coffee hour	Feedback and assignment Q&A session
Dec 22TH 08:45	Lecture	Advanced Repository Mining
Dec 23RD 23:59	Deadline	3RD deadline for Design a Study
Jan 10TH 13:30	Coffee hour	Feedback and assignment Q&A session
Jan 12TH 08:45	Lecture	Threats to Validity: Trustworthiness
Jan 13TH 23:59	Deadline	2ND deadline for Describe a Study
Jan 17TH 13:30	Workshop	Recap & Threats to Validity
Jan 19TH 08:45	No lecture	
Jan 31ST 09:00	Exam	Final examination
Feb 3RD 23:59	Deadline	Final deadline for Design a Study
Apr 12TH 13:30	Exam	Resit

3.3 Lectures

During the first lecture of *empirical methods in software engineering*, we introduce the course structure and discuss the assignments and deadlines. This helps clear up any confusion students might have about the course and the assignments and ensures that students know what is expected of them. Next, we introduce the fundamental differences between empiricism and rationalism (based on the article of Ralph [44]) and discuss several examples of empirical software engineering papers [28, 33, 37, 43, 68]. We have chosen these papers because they show a broad spectrum of software engineering problems being addressed and a wide variety of data collection and analysis methods employed. This introductory lecture is followed by a lecture on research questions and research strategies, based on the classical chapter by Easterbrook et al. [13] and the article of Stol and Fitzgerald [53].

The second week starts with a guest lecture, in which guests from industry discuss how they apply empirical research methods. To prevent guest speakers from going out of scope, we generally tend to invite contacts that we already know; additionally, we clearly communicate the goal of the guest lecturer to the speakers. The strategy of Eindhoven University of Technology for 2030 mentions a need for the practical application of theoretical knowledge. This is why we use the guest lectures to explain to students that applying empirical methods is not limited to "the ivory tower of academia." In 2022, we allocated 2 hours for a guest lecture, but in 2023, we decided to shorten it to 1 hour to make space for a lecture on design science. Indeed, in 2022, we have observed that while the lion's share of the course was dedicated to providing students with a means of understanding software engineering phenomena, in their follow-up master projects, they are often tasked with designing tools and evaluating their performance. To address the needs related to designing tools and to help students reason about the novelty and relevance of their work, we included a 1-hour lecture on design science based on the article by Engström et al. [15]. In the second week, we also teach a lecture on the topic of sampling following Baltes and Ralph [3].

Lectures on the third and fourth weeks are dedicated to data collection techniques: interviews and surveys, based on such papers as the work of Seaman [49], Hove and Anda [24], and Strandberg [54], and mining software repositories including discussion of the "promises and perils" associated with repository mining [6, 27]. Lectures on the fifth and the sixth weeks focus on data analysis: quantitative, to bring students up to speed with such statistical methods as regression discontinuity design [26] and mixed-effects modeling [4]; qualitative, to make the students aware that (post-)positivism is not the only valid paradigm in software engineering research based on the discussion by Melegati and Wang [36] and introduce the basic notions related to grounded theory following Hoda [22]; as well as on specific issues related to analysis of data from software repositories.

The last lectures cover the topics related to threats to validity following Wohlin et al. [63] and trustworthiness in qualitative studies following Lincoln and Guba [35]. Finally, the course is concluded with a recapitulation lesson.

3.4 Workshops

The first edition of the course (2020) consisted solely of lectures. Due to the COVID-19 pandemic, the lectures have been video recorded, which is why we experimented with a flipped classroom design in 2021. In this design, we requested that students watch the recorded lectures at home, and we organized workshops for every teaching session. This was not appreciated by the students, as they missed a more traditional way of transferring knowledge. The recorded lectures were ill-suited as a sole way of supporting self-study, and organizing two workshops per week has incurred an enormous amount of work for us. Additionally, we found that not all topics were equally suitable for 90-minute workshops: e.g., in the workshop on interviews, we found that 90 minutes is simply too brief to let students practice with the drafting and practicing of semi-structured interviews. This is why, from 2022 onward, we have decided to revert to the more traditional lecture-based setting, allocating three teaching sessions to workshops. As shown in Table 1, the workshops are dedicated to reading empirical papers, mining software repositories, and threats to validity.

3.4.1 Workshop: Reading Empirical Papers

During the reading workshop, the students are introduced to academic publishing (e.g., publication venues and structure of an empirical paper). Then, they are given 15 minutes to scan an empirical paper assigned to them and prepare a 2-minute-long pitch in which they answer the following questions:

- What problems do the authors try to address?
- Who benefits from the research of the authors?
- What strategy following Stol and Fitzgerald [53] do they use?
- What research methods in the strategy do they use?
- Is this paper a design science paper? If so, what is the design science component? (2023)
- What might have invalidated the results of the authors?
- What did the authors find?

For this task, we used papers by Moldon et al. [40], Vidoni [58], Wan et al. [60], and Danilova et al. [11]. The students then pitch the papers to each other and compare their insights. We try to facilitate a class discussion after each pitch, so we ask the students who also read the paper to respond to the pitch.

In the second part of the workshop, we introduce students to the idea that a single published paper might report the results of two or more individual empirical studies. Therefore, we ask the students to read a different paper (either by Begel and Zimmermann [5] or by Ralph et al. [45]) in more detail and indicate how many individual studies the paper describes. For each study, we then ask them to identify the research question, the methodology, the sampling strategy used, and the

conclusions of the studies. The students are asked to discuss their findings with their neighbors.

3.4.2 Workshop: Mining Software Repositories

The workshop commences after a lecture on mining software repositories (MSR). The workshop aims to teach students how to apply the material from the lecture on MSR (promises and perils) to a specific instance of a dataset. Additionally, we use this workshop to recapitulate the material on drafting and evaluating research questions, as this is something that students need for their assignment, *Design a Study*, and are known to have difficulties with. The learning goals of the workshop are the following:

- Students are able to formulate research questions based on a dataset and a research gap.
- Students are able to evaluate research questions with respect to feasibility and motivation.
- Students are able to apply the material of MSR to collect data from a software engineering dataset.

The workshop opens with a brief slide deck to recapitulate relevant theory for the workshop and to introduce the dataset (Stack Exchange data explorer[9]). Note that while the lecture on MSR covered the platforms Git and GitHub, the workshop uses Stack Exchange. We purposefully chose Stack Exchange such that students have to reason about a new platform. In the presentation, we also cover a research gap that students will work on during the workshop. Additionally, students are generally familiar with Stack Overflow, and we expect them to have some familiarity with SQL.

Table 2 lists the lesson plan for the workshop. In the workshop, we purposefully apply Rosenshine's principles of instruction [47]. When opening the workshop, we ground the material of the workshop by recapitulating the material of the lecture on MSR, and we explain how the material of this workshop relates to the research strategies of Stol and Fitzgerald [53] discussed in Lecture 2. Furthermore, we alternate the periods where a teacher is talking with periods where the students are engaging with the material themselves.

3.4.3 Workshop: Threats to Validity

During the threats to validity workshop, we let students draft and discuss threats to validity according to the model of Wohlin [63]. We start the workshop by briefly recapitulating the theory on threats to validity. After recapitulating, we give students

[9] https://data.stackexchange.com/.

Table 2 Lesson plan for the workshop on mining software repositories

Time (min)	Topic	Resource	Content
5	Recap	Slides	Welcome students, open the lecture. Start by recapitulating the theory on research strategies and the lecture on MSR. Explain how the material of this workshop allows students to apply this material
10	Gap & Tool	Slides	Briefly summarize two studies that use Stack Overflow as dataset, summarize the research gap introduced in these two papers, and introduce the publicly available Stack Overflow dataset students will use for the remainder of the workshop
5	Think	–	Ask students to draft research questions based on the research gap individually. Remind them of the material on research strategies and how to draft research questions
5	Share	Excel	Ask students to share their research questions with their neighbors and evaluate their partner's research question concerning motivation and relevance. Finally, ask some pairs to share their research questions with the group, and give feedback on them so that students can see how I evaluate research questions. Finally, ask students to share their research questions in the shared Excel sheet
5	Tips	Slides	Give students a set of tips and best practices for conducting MSR research. Tell them that for the rest of the workshop they will work on answering their drafted research questions using the Stack Overflow dataset
15	Work	–	Students work on their research questions, and a teacher walks around to ask pairs of students how they are progressing and if they have any questions
15	Break	–	–
20	Work	–	Students work on their research questions. Where needed, a teacher supports students
5	Update	–	Ask students who have already obtained results to share their results with the class and discuss their approaches and their queries. Ask students to reflect on their approaches and whether and how they have followed the guidelines
15	Work	–	Students work on their research questions. We try to support those students who are having more trouble with the material so that at the end of the workshop, hopefully, everyone has some results
5	Wrap-up	Slides	Wrap up the workshop, summarize, and explain how students can use the material of this lecture for their assignment. Announce the topic of the next lecture

a small interactive exercise: We show them a snippet in which a threat to validity is discussed, and we ask them whether the threat to validity was a threat to *Internal*, *External*, *Construct*, or *Conclusion* validity. Afterward, we host a small discussion in which we ask them to explain their answers and discuss their reasoning. We use this moment to reiterate how different threats to validity are distinguished from each other and their definitions.

Continuing on, we briefly explain some common pitfalls in threats to validity. For instance, we explain to students that threats to validity are not an excuse to do nothing: While working on their projects, they cannot make faulty choices and describe them as threats to validity.

During the final 45 minutes, we end with one large, practical exercise. We give students the paper of Kinsman et al. [29] in which we redacted the threats to validity section. We ask students to read the paper, identify the studies that are part of it, and then identify any threats to validity present. In a way, this exercise is the culmination of most of the knowledge we expect students to master: They need to read a paper, identify the methodologies used within it, and discuss any relevant shortcomings of the authors that might invalidate or threaten conclusions. Furthermore, the paper of Kinsman et al. combines both qualitative and quantitative methods, giving us a chance to discuss a wide range of threats to validity. Usually, while students are reading the paper, we walk around and engage in conversation with them. These one-on-one conversations allow us to guide students in their thinking and discuss their reasoning in a bit more detail. Finally, we wrap up the workshop with a brief central discussion; we invite students to share their threats and encourage them to give feedback on each other's threats.

3.5 Assessment

Similar to other courses at our university, we opt for a combination of a written exam and homework assignments. The final grade for the course is the weighted average of the *assignment grade* (70%) and the exam (30%), see Table 3. Additionally, to pass the course, an individual student's exam and assignment grades should both be 5.0 or higher. This requirement ensures that students who fail either the exam or both assignments cannot pass the course. The *assignment grade* is the weighted average of the Design a Study assignment (70%) and the Describe a Study assignment (30%).

Table 3 Weighting of each assessment moment and the learning objectives each assessment covers

Assessment	LO1	LO2	LO3	LO4	LO5	Weight
Design a study	10%	28%	7%		5%	50%
Describe a study				20%		20%
Exam		12%	12%		8%	30%
Total	10%	40%	19%	20%	13%	100%

3.5.1 Design a Study

Design a Study is a group assignment designed for groups of four students. During Design a Study, students are asked to take an existing dataset and use it to conduct an empirical study. We have paid special attention to selecting papers with dataset descriptions. We let students choose between different kinds of data, e.g., data about bugs and data about developer communication because we want to reduce the risk of plagiarism and give students the freedom to choose a dataset they think is interesting. The writing should be easily accessible for students with limited software engineering knowledge, the data should be stored either as comma-separated values or as a relational database, and the dataset should not be too large. The screening of datasets allows our students to focus more on applying the course material, as in 2021, unfortunately, working with a large No-SQL database was too challenging for many students. Students spent much time and effort trying to master the No-SQL query language and not on empirical research.

In 2022, we have suggested the students the following MSR 2022 datasets:

- A Large-scale Dataset of (Open Source) License Text Variants [67][10]
- An Alternative Issue Tracking Dataset of Public Jira Repositories [41][11]
- A Time Series-Based Dataset of Open-Source Software Evolution [50][12]
- A Versatile Dataset of Agile Open Source Software Projects [56][13]
- DaSEA—A Dataset for Software Ecosystem Analysis [7][14]
- DISCO: A Dataset of Discord Chat Conversations for Software Engineering Research [55][15]
- The OCEAN mailing list data set: Network analysis spanning mailing lists and code repositories [61][16]

Each year, we replace some of the datasets, based on the popularity of the datasets among the students. For the academic year 2023, we replaced the first two datasets with:

- A Dataset of Bot and Human Activities in GitHub [9][17]
- GitHub OSS Governance File Dataset [65][18]
- GIRT-DATA: Sampling GitHub Issue Report Templates [42][19]

[10] https://hal.archives-ouvertes.fr/hal-03624198/document.

[11] https://arxiv.org/pdf/2201.08368.pdf.

[12] https://drive.google.com/file/d/1svIexRcWrXBhb_pCVhfs7yKRzGsyaBpV/view.

[13] https://solar.cs.ucl.ac.uk/pdf/tawosi2022msr.pdf.

[14] https://itu.dk/~ropf/blog/assets/msr2022.pdf.

[15] http://olgabaysal.com/pdf/MuthuSubash_MSR2022_DataShowcase.pdf.

[16] https://arxiv.org/pdf/2204.00603.pdf.

[17] https://decan.lexpage.net/files/MSR-2023.pdf.

[18] https://zenodo.org/records/7530768.

[19] https://arxiv.org/pdf/2303.09236.pdf.

In this assignment, students are tasked with designing novel research questions, motivating and contextualizing their research questions, positioning their work with respect to the existing literature, designing and describing an appropriate methodology, reporting the results, and reasoning about the validity and trustworthiness of their work. With this assignment, we aim to cover the learning objectives of *empirical methods in software engineering* that are related to the design, execution, and evaluation of an empirical study (**LO1**, **LO2**, **LO3**).

In the first editions of the course, we observed that students had difficulties coming up with good research questions or picking the right methodology. This is why we have decided to split the assignment into four parts with four different deadlines: students are encouraged but not required to submit partial versions of the report by the first three deadlines. We provide them with feedback on the novelty of the research questions, the feasibility of the proposed study, the writing, the relevance of the research questions, the motivation of the work, and the methodology. Where possible, we try to give our feedback in the form of open, reflective questions. We do not grade the submissions: this allows students to make mistakes, learn from these mistakes without being penalized, and iteratively improve their work. Since the same assessment criteria are used to grade the final submission, the students get an early indication of their learning progress.

Specifically, by the first deadline, the students should select three of the above dataset papers, summarize each paper in a single paragraph, and propose four research questions that one can answer based on one (or more) of the datasets. By the second deadline, the student should submit a draft of their final report, including an introduction, related work, and methodology. The study design should address two research questions based on one or two of the datasets. By the third deadline, the students should revise the report based on the feedback received so far and sketch the study results. The fourth and final deadline of the assignment is the full submission of the report, including results, discussion, threats to validity, and conclusions. Since *Design a Study* is a complex assignment, the final deadline is at the very end of the course, after the exam (see Table 1). In this way, we create a gap of 4 weeks between the last formative and final deadlines. This ensures that the students have 4 weeks to work on the assignment, implement our feedback, and finalize their report.

3.5.2 Describe a Study

Describe a Study is an individual assignment. The goal of Describe a Study is to teach students how to accurately communicate the findings from the academic literature in a way that is accessible to a broad audience. Graduates will likely be working in teams where not everyone is familiar with reading and disseminating academic literature, and they should be able to build bridges between scientific research and industrial practice. The assignment covers **LO4**; it was inspired by

the "It Will Never Work in Theory" project of Greg Wilson[20] and the Practitioners' Digest initiated by IEEE Software in 2015 [51].

Since students are usually not familiar with writing a summary of a paper for a broader audience, in 2022, we have chosen to have two iterations of the assignment. For each iteration, we have selected several recently published empirical software engineering studies that do not explicitly list implications for practitioners. We restricted the number of students summarizing the same paper to seven; we have offered a choice of 11 papers for the first iteration [1, 2, 10, 14, 18, 20, 23, 30, 31, 59, 66] and 7 papers for the second iteration [12, 17, 19, 25, 34, 38, 64]. Like Design a Study, the first iteration is not mandatory or graded; instead, we provide feedback to help you improve the students' writing. For the second iteration, the students were requested to submit a summary of a different paper.

The feedback we have received from the students has indicated, however, that their chances to improve the summary were limited due to switching to a different paper in the second iteration. This is why, in 2023, while keeping the two-iteration principle, we adjusted the assignment so that students could work on the same scientific paper for both iterations. Specifically, for *the first iteration*, the students select a paper and write a summary or script for the two-minute-long video pitch. The students are expected to remember that practitioners are not familiar with academic research or the methodologies used by software engineering researchers. Furthermore, the students should be aware that while researchers are interested in the exact methodology, practitioners can be expected to care less about how a study was conducted but instead about what the study means for their work. For *the second iteration*, the students record the pitch as if they have been invited to speak at a large developer conference. To record the video, the students are free to use any tool they like; however, in the video, the students' faces should be visible. Additionally, the students are free to use any supporting material they like, such as slides or images; however, these should not distract too much from the pitch itself.

The grading criteria for the pitch are the quality of the content and delivery as well as its relevance for practitioners:

- *Content*: Does the pitch contain a clear and accurate summary of the research paper? Does the pitch contain a clear description of the findings of the paper?
- *Delivery*: Is the pitch delivered in a clear and concise manner? Is the pitch delivered in a way that is engaging for the audience? Does the pitch stay within the time limit?
- *Relevance*: Does your pitch highlight the relevance of the research for practitioners? Does it accurately address the question: Why should practitioners care about this research?

[20] https://neverworkintheory.org/category/.

3.5.3 Exam

The 2021 exam consisted mostly of multiple-choice questions to ensure that grading of the exam could be done efficiently. However, after speaking to students at the end of the exam and inspecting the grades, we found out that even good students sometimes understood the material but interpreted questions differently than intended. Moreover, by definition, such an exam provides very limited insights into the way students reason about a study design.

Hence, since the 2022 edition of *empirical methods in software engineering*, we opt for a 90-minute closed book exam. The exam covers learning objectives **LO3** and **LO5** and consists of a mix of open and multiple-choice questions. Five multiple-choice questions (4 points each) are intended to check the understanding of the basic notions of empirical software engineering research (learning outcome **LO5**): e.g., one of the questions asked the students to identify which of the following aims would justify conducting a laboratory experiment: building a new theory based on the existing ones, establishing causal relations, gaining profound understanding of the study participants' experiences or opinions, or understanding the phenomenon under study in the most realistic context. Next, we sketch the context of a scientific study and a corresponding research question and ask the students to design a study methodology that would allow a researcher to answer the research question. To help the students, we ask a series of sub-questions related to the study design, i.e., motivate the choice for the research strategy among those described by Stol and Fitzgerald [53], describe a data collection method, describe and motivate the sampling approach, and describe and motivate the data analysis approach. In total, this group of questions amounts to 40 points. For the final part of the exam, we provide the students with a four-page empirical software engineering paper with the threats to validity section censored. The students should read the paper and propose two viable and distinct threats to validity. For each threat to validity, they should explain which conclusions of the study would be invalidated by the threat and how and classify the threat as a threat to construct, internal, external, or conclusion validity. The threats described should belong to two different types of threats to validity, i.e., if the first threat to validity belongs to the category "external," the second threat you describe cannot belong to the category "external." This part of the exam is also scored on a scale of 40 (9 points for the description of each one of the threats, 9 points for the explanation of what conclusions are invalidated and how, and 2 points for the classification of the threats in the corresponding category).

3.6 Limitations and Implications

The choices made in the course design described in this section were partly made based on the following limitations:

- *Course duration*: Because of the 10-week-long teaching period at Eindhoven, there is not enough time to have students write a *publishable* scientific study during the course. While we would consider this to be the most appropriate practical assignment for the course, it is unfortunately infeasible given the very short timeline of the course: The Design a Study reports written by students often focus on less-relevant research questions, and because of the 10-week teaching period, there is no time to give students enough time to rework their own research questions. This being said, we have co-authored an MSR mining challenge paper [39] with a group of students from the 2021 edition of *empirical methods in software engineering* based on their Design a Study report. Unfortunately, this is no longer an option for more recent course editions due to the MSR mining challenge deadlines being moved from February to December and the course being taught from mid-November to early February.
- *Technological knowledge*: Students participating in *empirical methods in software engineering* course are diverse with respect to their study program and prior knowledge. Because of this, some students are experienced programmers and data analysts. However, there are also students who have very little programming experience, and this limits the topics we can cover during more technical workshops, such as the one on MSR. Additionally, in one of the previous editions, we used a dataset stored in MongoDB: querying a non-relational database turned out to be too difficult for some of the students, hindering their ability to ask queries interesting from the software engineering perspective. Therefore, we have discontinued using this dataset, and instead, we let students pick from a more diverse set of datasets for the *Design a Study* assignment.
- *Available resources*: It is not ideal to use a closed-book exam to test the course's high-level learning objectives. Preferably, we would have assessed students' ability to apply empirical methods to study software engineering through a small research project, similar to the design a study assignment. However, the rules of Eindhoven University of Technology require each course to contain an individual assessment. Therefore, grading cannot be based entirely on group projects. To include individual assessments, we could, of course, make the design of a study assignment individual. However, because *empirical methods in software engineering* is taught by the two authors of this chapter, it is unfeasible for them to give feedback on, and grade, individual assignments preserving an inherent diversity of individual research projects. Therefore, we combine the group project with an individual exam.
- *Ethics*: Ideally, we would allow students to use the full set of research methods discussed in the lectures, and this includes research involving humans such as interviews or surveys. However, research with humans requires ethical approval from the Eindhoven ethics review board. Because of the course's limited time frame and the ethics review board's processing time, it is impossible to require students to obtain permission before executing their study.

4 Grades and Student Feedback

In this section, we focus on the student assessment results, i.e., grades for Design a Study, Describe a Study, the exam, and the final course grade, as well as on the course evaluation by students. While the former is intended to assess the quality of learning, the latter is intended to assess the quality of teaching, providing complementary perspectives on *empirical methods in software engineering*. However, we know the limitations [57] and biases [21, 32] inherent in student evaluations of teaching. This is why we complement the formal student evaluation organized by the university with the feedback we have obtained from students during the course run through informal chats, dedicated feedback moments, anonymous feedback forms, and feedback obtained from colleagues. We focus on the 2022 edition of the course and its exam that took place in January 2023, since the exam for the 2023 edition is scheduled for January 2024 and has not yet taken place at the time of writing.

4.1 Grades

4.1.1 Describe a Study and Design a Study

Forty-nine students have submitted the "Describe a study" assignment: four students have received less than 5.0, one student has received the perfect score of 10, and the median score was 7.1.

While working on the "Design a study" assignment, students have been encouraged to work in groups of four. However, we did not want to enforce collaborations, and, hence some groups were smaller. In the beginning of the course, we had 9 four-student groups, 3 three-student groups, 3 two-student groups, and 1 group consisting of a single student. Students from the smaller groups have been warned that the effort required to complete the assignment has been designed for four students and that the grading criteria will be applied to the submissions independently from the number of group members. At the end, 45 students have submitted the assignment: all students have passed the threshold of 5.0, and the median and the mean scores are 6.8.

"Describe a study" and "Design a study" have been designed to cover different learning objectives: **LO4** for "Describe a study" and **LO1**, **LO2**, **LO3**, and **LO5** for "Design a Study." This is why we expect the grades of the "Describe a study" and "Design a study" not to exhibit a strong correlation. This is indeed the case: Pearson's $r \simeq 0.015$, Spearman's $\rho \simeq= -0.1$, and none of these is statistically significant, suggesting that the two assignments indeed assess complementary aspects of the students' knowledge.

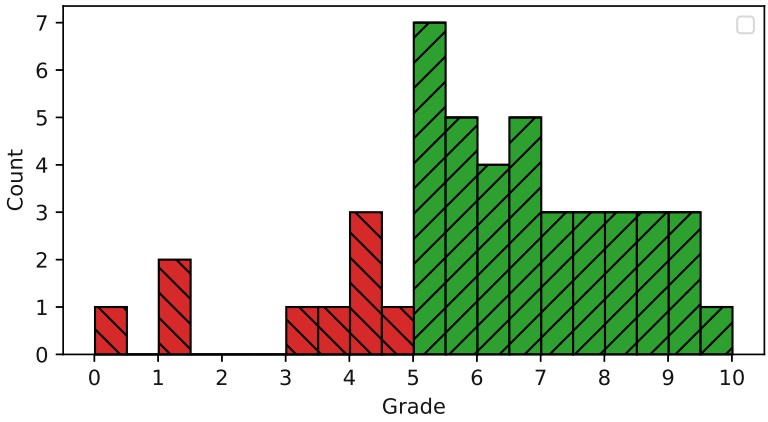


Fig. 1 Histogram of the grade distribution for the exam

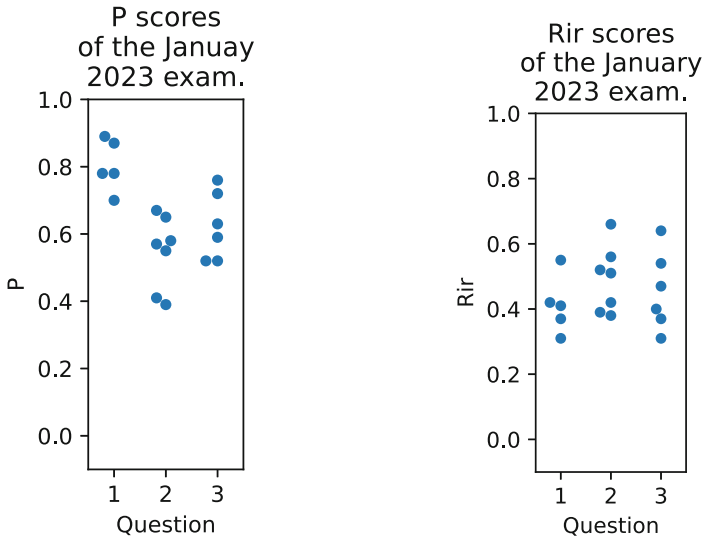
4.1.2 Exam

Forty-six students have participated in the first exam attempt of the 2022 edition of *empirical methods in software engineering*. Three students have submitted empty or near-empty exam papers and were clearly unprepared for the exam. Six students have submitted a non-empty exam paper that could be seen as an attempt to pass the exam but got a grade lower than 5.0. Seven students obtained a grade between 5.0 and 5.5: normally speaking, this grade would be seen as insufficient, but as explained in Sect. 3.5, with a high-enough assignment grade, these students can still pass the course. Finally, the remaining 30 students have passed the exam (Fig. 1).

To understand the quality of the individual exam questions, we have used the analysis metrics included in ANS Delft, the assessment platform provided by our university.[21] ANS Delft includes several such metrics; we opt for P and rir scores. The P-score is a measure of the difficulty of the question. It is defined as the fraction of the participants who answered the question successfully. Meanwhile, the rir score computes the correlation between the grade of the question and the exam as a whole, with higher values indicating that students who performed well on the question tend to receive a better grade. The assumption underlying rir is that overall, students performing well on the exam as the whole are expected to perform well on individual questions as well.

Figure 2 summarizes the P-scores and rir score of the exam. Each dot corresponds to the score of a subquestion and question to part of the exam. For example, the five dots corresponding to Question 1 in Fig. 2a show P scores of five multiple-choice questions designed to check the understanding of the basic notions of empirical software engineering (see Sect. 3.5.3).

[21] https://ans.app/landing.



(a) P-values as computed by ANS Delft

(b) Rir values as computed by ANS Delft

Fig. 2 Quality of individual exam questions

Figure 2a shows that the P-scores are almost always higher than 0.4 There is only one P-score lower than 0.4, and its value is 0.39. The scores are higher for the subquestions of the first question than for the subquestions of the second and third questions. This is to be expected, as the closed subquestions for the first question are meant to be easier than the open subquestions for the second and third questions. Meanwhile, the second and third questions are meant to be harder to answer, so the lower p-scores for these subquestions are expected and not immediately problematic. However, there are two subquestions for the second question that only 40% of the students could answer correctly, which might be a sign to review these questions for future exams.

The rir scores are plotted in Fig. 2b. There are no outliers, and most importantly, the rir values are distributed equally across the different questions. The median value is slightly higher than 0.4, indicating that the outcome of most questions correlates with the outcome of the exam as a whole. Overall, rir scores higher than 0.4 are considered to be quite good.[22]

The high values for both metrics considered indicate that the exam is **reliable**.

[22] https://support.ans.app/hc/en-us/articles/360027234814-How-to-analyse-questions.

4.1.3 Course Grades

Out of 59 students registered for the course, 36 passed it, i.e., scored 6 or higher. Unlike the assignment grades or exam grades, course grades are integers. 8 students scored 6, 17 students scored 7, 9 students scored 8, and 2 students scored 9. Among the remaining students, 16 did not complete the course requirements (i.e., did not submit homework assignments or did not participate in the exam).

4.2 Feedback

4.2.1 Informal Feedback

As stated above, we complement the formal student evaluation organized by the university (see Sect. 4.2.2) with the feedback we have obtained from students during the course run by means of informal chats, dedicated feedback moments, and anonymous feedback form, as well as feedback obtained from colleagues.

The set of concerns raised by students in the anonymous feedback form is very diverse. Students use it to ask questions like whether we can upload slides to lectures before the lecture itself. However, they also use the feedback form to express their concerns: In both 2022/2023 and 2023/2024, students wrote that they were unsure about how to prepare for the exam. Students indicated that they found the lectures and primary material very abstract and theoretical and unsure how to answer the exam questions. We believe that the gap between theory and application of the theory is one of the challenges of teaching this empirical methods course: the currently available material is theoretical, and in a 10-week course, it is challenging to give students enough opportunities to apply the material.

4.2.2 Formal Student Evaluation

Thirteen students have filled in the student evaluation form sent by the faculty. While this corresponds to a response rate of 22%, this response ratio is similar to response rates for other courses. Overall, the students were positive: the course scored 3.8 on a scale of 5 and the teachers 4.2 on a scale of 5. In particular, the students have agreed that the educational setup (e.g., structure, content, teaching/learning methods, level, and coherence) worked well and was suitable for this course; the course was well organized, and the course material was clear and motivated students to study. The students expected that the material of this course would be useful for their graduation project, enjoyed the industrial talks at the start of the course, and believed the workshops were a useful addition to the course. They further indicated that the lecturer explained the content clearly and comprehensively, while the instructor helped them master the subjects.

For us as teachers, the most valuable feedback was provided in the open questions—what they liked about the course and what could be improved. The students mostly appreciated the relevance of the course topics ("Very applicable for anyone's master project and therefore important knowledge to have") and their uniqueness in the educational offering of our computer science program, quality of the teaching materials ("The beautiful, clear slides," "Notes with the slides were useful," "I liked how the slides were organized. They were creative and clear in structure"), diversity of the assignments ("It's an overview of many things I have studied/worked with before, so I could review everything again," "I enjoyed the topics that were discussed. They helped me better understand and classify studies I have been reading for other subjects or for my master's thesis"), intermediate evaluation ("Iterative approach for submissions with opportunities for feedback along the way," "A lot of voluntary effort was put into intermediate feedback (ungraded assignments and coffee hours)"), and, in general, the teachers and the course atmosphere ("I liked specially the professors," "Great ability of instructors to generate real and impactful discussions, which helped me think critically about the topic," "Atmosphere in class was very nice; both lecturers were enthusiastic not only about the material but also about teaching it (not always a given in Uni) and interacting with the class. <...> the passion behind the course was definitely a big pro; I enjoyed attending the lectures"). Students indicated that the non-graded character of the intermediate assignments did not motivate them and suggested determining at least 10% of the grade based on the intermediate assignments. We, however, believe that making these assignments graded would prevent students from exploring (and making mistakes), which is crucial at early stages of learning. Another comment referred to the feedback from the Describe a Study assignment and students' inability to apply the feedback for the non-graded submission to the graded one. To address this comment in 2023, we have revised the Describe a Study assignment as described in Sect. 3.5.2. Furthermore, one of the students requested a video recording of the lectures: while the video recordings of one of the pandemic course editions are available on YouTube and the link to the recordings has been announced during the first lecture, we recognize that the quality of the videos is, of course, not professional. This is why we have requested a professional recording in 2023. Finally, the students have indicated having difficulties with formulating research questions. We recognize this challenge; however, this is an inherent challenge of conducting research, and learning how to formulate research questions is an important milestone in researchers' growth. This is why we believe that this is an important goal of our course. To support students' learning, we provide detailed feedback on their homework, as well as discuss their research questions during one of the coffee hours.

Based on the student evaluation, the education program management has awarded us the Excellent Course Evaluation certificate.

5 Lessons Learned and Didactic Challenges

The lessons we have learned are partly related to the body of empirical software engineering knowledge and partially to the specifics of our educational setting.

5.1 Empirical Software Engineering Body of Knowledge

While preparing for this course, we have observed that multiple articles are available on the application of individual research methods in the context of software engineering, e.g., interviews, surveys, or controlled experiments. However, few articles provide a coherent vision on empirical software engineering, and few articles are trying to focus on the steps preceding the choice of an individual research method, namely, formulating research questions and understanding what research questions can be answered by means of an empirical study.

The chapter by Easterbroook et al. [13] makes an important step in supporting the learners in understanding how research questions can be formulated; however, having a well-formulated research question is not enough. A research question should also be *relevant* for software engineering practice, research, or education, and it should be *novel* with respect to the existing body of knowledge. Additionally, when formulating research questions, we observed that some students focus very strongly on the data available in the datasets and formulate research questions that can be answered based on the datasets but lose sight of relevance or novelty. For example, focusing on the dataset of communication in Discord communities of Python, Go, Racket, and Clojure [55], students suggest studying "how often programming languages are different than the one corresponding to the channel, mentioned in a channel." However, the report submitted did not indicate what stakeholder would benefit from answering this question and in what way(s). Based on the feedback, the students decided to revise their question. To support students in reflecting on the relevance and novelty of their research, we have introduced them to the design science framework of Engström et al. [15], but this remains a challenge we will need to continue working in the years to come.

Another challenge we have observed is the absence of a coherent delineation of empirical research. While all the sources seem to agree that empirical research should be based on observations, the further conceptualizations of this idea seem to diverge. For example, the ABC framework [53] and the comparison of rationalism and empiricism by Ralph [44] focus on studies seeking knowledge, implicitly equating these notions. Later on, Stol and Fitzgerald [52] (following Wieringa [62]) juxtapose knowledge-seeking and solution-seeking research. While both knowledge-seeking and solution-seeking research can be empirical, the ABC framework would be appropriate for the former, framework of design science as discussed by Runeson et al. [48] and Engström et al. [15] would be appropriate for the latter. However, the design science framework also includes studies focusing

solely on problem understanding, even though these studies seek knowledge rather than solutions. Moreover, the ABC framework of empirical research also covers *non-empirical* studies such as formal theory and computer simulation. This plurality of opinions and conceptualizations reflects the active quest for understanding inherent to scientific research, but at the same time, it makes it extra challenging to present the students with a coherent vision of empirical software engineering research.

5.2 Didactic Challenges

The next group of challenges is related to students being inexperienced researchers.

Conducting an empirical research study is by no means trivial, and teaching research skills requires a lot of scaffolding. We have provided significant support by offering intermediate ungraded assignments, providing datasets, and allocating dedicated sessions for students to ask questions beyond the regular teacher-student communication. We would recommend working with students and giving them feedback throughout the course for anyone who wants to teach empirical methods to students.

Next, we observed that students had a hard time using or acquiring technical skills related to analyzing data. For example, while SQL is being taught in all computer science and data science bachelor programs around the world and students are expected to have learned at some point during their studies, using this knowledge to answer questions about Stack Overflow using the Stack Exchange Data Explorer[23] proved to be difficult. Working with no-SQL databases such as MongoDB proved even more difficult, and we had to exclude datasets stored in such databases. Because the course focuses on empirical methods and not data analysis skills, we address this in the course design itself: we do not require any technical affinity, and we make available datasets to students that are easy to access.

Yet another challenge was related to the speed of reading academic texts. Since, unfortunately, reading academic papers is not part of a traditional bachelor curriculum in computer science, for many students, *empirical methods in software engineering* was the first exposure to academic writing. This might be demotivating, as one of the students has indicated, “I am not used to research and reading papers as you are, no matter how much I practice it.” Moreover, English is not the first language of our students. As shown by Busby and Dahl, while on its own not being a native speaker does not necessarily reduce the reading speed of an academic text, a combination of not being a native speaker and operating in a parallel language context where both English and the local language are used might reduce the reading speed [8]. To address this challenge, we have explained to the students how empirical papers are structured and encouraged them to highlight the relevant parts

[23] https://data.stackexchange.com/.

of a paper when reading it. We have also consciously limited the mandatory reading to four-page papers.

Finally, we have observed that many students were afraid of working with qualitative analysis approaches. This complex phenomenon can partly be attributed to the strong positivist tradition in computer science in general and in fundamental courses taught in the bachelor of computer science program in particular. We are, of course, not the first ones to observe this—Richards has discussed her experiences with teaching qualitative research course to students with the positivist backgrounds, and our experiences to some extent echo hers [46]. Two important differences, however, make our settings different from the one discussed by Richards and simultaneously limit our ability to address the "qualitative analysis fear." First, qualitative analysis is merely a subject of slightly more than one lecture in a single course in the master degree. As such, it remains "the odd one out," and students do not get enough time and opportunity to learn more about qualitative research and better appreciate it. Second, while both of the authors have conducted qualitative research as part of their empirical software engineering work, both of us are trained as computer scientists and had to learn qualitative analysis by self-learning and in an engineering context rather than through an established academic curriculum in a scientific discipline with a long tradition of qualitative inquiry. We unfortunately have no practical recommendation on how to address this lesson learned. Given the limitations of the course and especially the time frame, there is no time to give students more hands-on experiences with qualitative research.

6 Resources

Both the "pandemic" video recordings of the 2020 edition and the classroom video recordings of the 2023 edition are publicly available. We also make slides available as pdfs, previous exams including answer keys, and the syllabus. These can be accessed through the repository with supplementary materials accompanying this book.[24]

7 Conclusions

In this chapter, we have described the *empirical methods in software engineering* course taught by us at Eindhoven University of Technology. We have aimed to provide a rich description of the challenges we have experienced and decisions taken to address them. We further elaborate on the structure of the course and assessment strategy we implemented and reflect upon the feedback received from

[24] https://zenodo.org/doi/10.5281/zenodo.11544897.

students and peers. To support colleagues teaching similar courses elsewhere, we share a plethora of teaching materials, from lecture recordings and slide decks to homework assignments and assessment rubrics. We are mindful that teaching at a different institution might require radically different approaches but hope that our materials and insights will be able to serve as stepping stones.

References

1. Alamin, M.A.A, Malakar, S., Uddin, G., Afroz, S., Haider, T.B., Iqbal, A.: An empirical study of developer discussions on low-code software development challenges. In: 18th IEEE/ACM International Conference on Mining Software Repositories, MSR 2021, Madrid, Spain, May 17–19, 2021, pp. 46–57. IEEE (2021)
2. Alfadel, M., Costa, D.E., Shihab, E., Mkhallalati, M.: On the use of dependabot security pull requests. In: 18th IEEE/ACM International Conference on Mining Software Repositories, MSR 2021, Madrid, Spain, May 17–19, 2021, pp. 254–265. IEEE (2021)
3. Baltes, S., Ralph, P.: Sampling in software engineering research: a critical review and guidelines. Empir. Softw. Eng. **27**(4), 94 (2022)
4. Bates, D., Mächler, M., Bolker, B., Walker, S.: Fitting linear mixed-effects models using lme4. J. Stat. Software **67**(1), 1–48 (2015)
5. Begel, A., Zimmermann, T.: Analyze this! 145 questions for data scientists in software engineering. In: Proceedings of the 36th International Conference on Software Engineering, ICSE 2014, -p. 12–23. Association for Computing Machinery, New York, NY, USA (2014)
6. Bird, C., Rigby, P.C., Barr, E.T., Hamilton, D.J., Germán, D.M., Devanbu, P.T.: The promises and perils of mining git. In: Godfrey, M.W., Whitehead, J. (eds.) Proceedings of the 6th International Working Conference on Mining Software Repositories, MSR 2009 (Co-located with ICSE), Vancouver, BC, Canada, May 16–17, 2009, Proceedings, pp. 1–10. IEEE Computer Society (2009)
7. Buchkova, P., Hinnerskov, J.H., Olsen, K., Pfeiffer, R.H.: Dasea: A dataset for software ecosystem analysis. In: Proceedings of the 19th International Conference on Mining Software Repositories, MSR '22, pp. 388–392. Association for Computing Machinery, New York, NY, USA (2022)
8. Busby, N.L., Dahl, A.: Reading rate of academic english texts: Comparing l1 and advanced l2 users in different language environments. Nordic J. English Stud. **20**(1), 36–61 (2021)
9. Chidambaram, N., Decan, A., Mens, T.: A dataset of bot and human activities in github. In: 2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR), pp. 465–469 (2023)
10. Ciniselli, M., Cooper, N., Pascarella, L., Poshyvanyk, D., Penta, M.D., Bavota, G.: An empirical study on the usage of BERT models for code completion. In: 18th IEEE/ACM International Conference on Mining Software Repositories, MSR 2021, Madrid, Spain, May 17–19, 2021, pp. 108–119. IEEE (2021)
11. Danilova, A., Naiakshina, A., Horstmann, S., Smith, M.: Do you really code? designing and evaluating screening questions for online surveys with programmers. In: Proceedings of the 43rd International Conference on Software Engineering, ICSE '21, pp. 537–548. IEEE Press (2021)
12. de Souza Santos, R.E., Ralph, P.: A grounded theory of coordination in remote-first and hybrid software teams. In: Proceedings of the 44th International Conference on Software Engineering, ICSE '22, pp. 25–35. Association for Computing Machinery, New York, NY, USA (2022)
13. Easterbrook, S., Singer, J., Storey, M.A., Damian, D.: Selecting Empirical Methods for Software Engineering Research, pp. 285–311. Springer London, London (2008)

14. Eng, K., Hindle, A.: Revisiting dockerfiles in open source software over time. In: 18th IEEE/ACM International Conference on Mining Software Repositories, MSR 2021, Madrid, Spain, May 17–19, 2021, pp. 449–459. IEEE (2021)
15. Engström, E., Storey, M.D., Runeson, P., Höst, M., Baldassarre, M.T.: How software engineering research aligns with design science: a review. Empir. Softw. Eng. **25**(4), 2630–2660 (2020)
16. European Commission, Sport Directorate-General for Education, Youth, and Culture: ECTS users' guide 2015. Publications Office of the European Union (2015)
17. Fang, H., Lamba, H., Herbsleb, J., Vasilescu, B.: "this is damn slick!" estimating the impact of tweets on open source project popularity and new contributors. In: 2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE), pp. 2116–2129 (2022)
18. Galappaththi, A., Nadi, S., Treude, C.: Does this apply to me? An empirical study of technical context in stack overflow. In: 19th IEEE/ACM International Conference on Mining Software Repositories, MSR 2022, Pittsburgh, PA, USA, May 23–24, 2022, pp. 23–34. ACM (2022)
19. Gallaba, K., Lamothe, M., McIntosh, S.: Lessons from eight years of operational data from a continuous integration service: An exploratory case study of circleci. In: Proceedings of the 44th International Conference on Software Engineering, ICSE '22, p. 1330–1342. Association for Computing Machinery, New York, NY, USA (2022)
20. Grotov, K., Titov, S., Sotnikov, V., Golubev, Y., Bryksin, T.: A large-scale comparison of python code in jupyter notebooks and scripts. In: 19th IEEE/ACM International Conference on Mining Software Repositories, MSR 2022, Pittsburgh, PA, USA, May 23–24, 2022, pp. 353–364. ACM (2022)
21. Heffernan, T.: Sexism, racism, prejudice, and bias: a literature review and synthesis of research surrounding student evaluations of courses and teaching. Assess. Evaluat. Higher Educ. **47**(1), 144–154 (2022)
22. Hoda, R.: Socio-technical grounded theory for software engineering. IEEE Trans. Software Eng. **48**(10), 3808–3832 (2022)
23. Hora, A.C.: Googling for software development: What developers search for and what they find. In: 18th IEEE/ACM International Conference on Mining Software Repositories, MSR 2021, Madrid, Spain, May 17–19, 2021, pp. 317–328. IEEE (2021)
24. Hove, S.E., Anda, B.: Experiences from conducting semi-structured interviews in empirical software engineering research. In: 11th IEEE International Symposium on Software Metrics (METRICS 2005), 19–22 September 2005, Como Italy, p. 23. IEEE Computer Society (2005)
25. Hu, X., Xia, X., Lo, D., Wan, Z., Chen, Q., Zimmermann, T.: Practitioners' expectations on automated code comment generation. In: Proceedings of the 44th International Conference on Software Engineering, ICSE '22, pp. 1693–1705. Association for Computing Machinery, New York, NY, USA (2022)
26. Imbens, G.W., Lemieux, T.: Regression discontinuity designs: A guide to practice. J. Economet. **142**(2), 615–635 (2008). The regression discontinuity design: Theory and applications
27. Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., Germán, D.M., Damian, D.E.: An in-depth study of the promises and perils of mining github. Empir. Softw. Eng. **21**(5), 2035–2071 (2016)
28. Khalid, H., Shihab, E., Nagappan, M., Hassan, A.E.: What do mobile app users complain about? IEEE Software **32**(3), 70–77 (2015)
29. Kinsman, T., Wessel, M., Gerosa, M.A., Treude, C.: How do software developers use github actions to automate their workflows? pp. 420–431. IEEE (2021)
30. Kinsman, T., Wessel, M.S., Gerosa, M.A., Treude, C.: How do software developers use github actions to automate their workflows? In: 18th IEEE/ACM International Conference on Mining Software Repositories, MSR 2021, Madrid, Spain, May 17–19, 2021, pp. 420–431. IEEE (2021)
31. Kochanthara, S., Dajsuren, Y., Cleophas, L., van den Brand, M.: Painting the landscape of automotive software in github. In: 19th IEEE/ACM International Conference on Mining Software Repositories, MSR 2022, Pittsburgh, PA, USA, May 23–24, 2022, pp. 215–226. ACM (2022)

32. Kreitzer, R.J., Sweet-Cushman, J.: Evaluating student evaluations of teaching: a review of measurement and equity bias in sets and recommendations for ethical reform. J. Acad. Ethics **20**(1), 73–84 (2022)
33. Krueger, R., Huang, Y., Liu, X., Santander, T., Weimer, W., Leach, K.: Neurological divide: An fmri study of prose and code writing. In: IEEE/ACM 42nd International Conference on Software Engineering (ICSE), pp. 678–690 (2020)
34. Liang, J.T., Zimmermann, T., Ford, D.: Understanding skills for oss communities on github. In: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2022, pp. 170–182. Association for Computing Machinery, New York, NY, USA (2022)
35. Lincoln, Y.S., Guba, E.G.: Naturalistic Inquiry. Sage Publications (1985)
36. Melegati, J., Wang, X.: Surfacing paradigms underneath research on human and social aspects of software engineering. In: IEEE/ACM 13th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE), pp. 41–50 (2021)
37. Meyer, A.N., Barr, E.T., Bird, C., Zimmermann, T.: Today was a good day: The daily life of software developers. IEEE Trans. Software Eng. **47**(5), 863–880 (2021)
38. Miller, C., Cohen, S., Klug, D., Vasilescu, B., KaUstner, C.: "did you miss my comment or what?": Understanding toxicity in open source discussions. In: Proceedings of the 44th International Conference on Software Engineering, ICSE '22, p. 710–722. Association for Computing Machinery, New York, NY, USA (2022)
39. Moharil, A., Orlov, D., Jameel, S., Trouwen, T., Cassee, N., Serebrenik, A.: Between JIRA and github: Asfbot and its influence on human comments in issue trackers. In: 19th IEEE/ACM International Conference on Mining Software Repositories, MSR 2022, Pittsburgh, PA, USA, May 23–24, 2022, pp. 112–116. ACM (2022)
40. Moldon, L., Strohmaier, M., Wachs, J.: How gamification affects software developers: Cautionary evidence from a natural experiment on github. In: Proceedings of the 43rd International Conference on Software Engineering, ICSE '21, pp. 549–561. IEEE Press (2021)
41. Montgomery, L., Lüders, C., Maalej, W.: An alternative issue tracking dataset of public jira repositories. In: Proceedings of the 19th International Conference on Mining Software Repositories, MSR '22, pp. 73–77. Association for Computing Machinery, New York, NY, USA (2022)
42. Nikeghbal, N., Kargaran, A.H., Heydarnoori, A., Schutze, H.: Girt-data: Sampling github issue report templates. In: 2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR), pp. 104–108. IEEE Computer Society, Los Alamitos, CA, USA (2023)
43. Qiu, H.S., Nolte, A., Brown, A., Serebrenik, A., Vasilescu, B.: Going farther together: The impact of social capital on sustained participation in open source. In: IEEE/ACM 41st International Conference on Software Engineering (ICSE), pp. 688–699 (2019)
44. Ralph, P.: The two paradigms of software development research. Sci. Comput. Programm. **156**, 68–89 (2018)
45. Ralph, P., Baltes, S., Adisaputri, G., Torkar, R., Kovalenko, V., Kalinowski, M., Novielli, N., Yoo, S., Devroey, X., Tan, X., Zhou, M., Turhan, B., Hoda, R., Hata, H., Robles, G., Milani Fard, A., Alkadhi, R.: Pandemic programming: How covid-19 affects software developers and how their organizations can help. Empirical Softw. Engg. **25**(6), 4927–4961 (2020)
46. Richards, J.C.: "every word is true": Stories of our experiences in a qualitative research course. Qualitat. Rep. **16**(3), 782–819 (2011)
47. Rosenshine, B., Stevens, R.: Teaching functions. Handbook Res. Teach. **3**, 376–391 (1986)
48. Runeson, P., Engström, E., Storey, M.D.: The design science paradigm as a frame for empirical software engineering. In: Felderer, M., Travassos, G.H. (eds.) Contemporary Empirical Methods in Software Engineering, pp. 127–147. Springer (2020)
49. Seaman, C.B.: Qualitative methods. In: Shull, F., Singer, J., Sjøberg, D.I.K. (eds.) Guide to Advanced Empirical Software Engineering, pp. 35–62. Springer (2008)

50. Sousa, B.L., Bigonha, M.A.S., Ferreira, K.A.M., Franco, G.C.: A time series-based dataset of open-source software evolution. In: Proceedings of the 19th International Conference on Mining Software Repositories, MSR '22, pp. 702–706. Association for Computing Machinery, New York, NY, USA (2022)
51. Spinellis, D.: Introductions. IEEE Software **32**(4), 3–5 (2015)
52. Stol, K., Fitzgerald, B.: Guidelines for conducting software engineering research. In: Felderer, M., Travassos, G.H. (eds.) Contemporary Empirical Methods in Software Engineering, pp. 27–62. Springer (2020)
53. Stol, K.J., Fitzgerald, B.: The abc of software engineering research. ACM Trans. Softw. Eng. Methodol. **27**(3) (2018)
54. Strandberg, P.E.: Ethical interviews in software engineering. In: 2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM 2019, Porto de Galinhas, Recife, Brazil, September 19-20, 2019, pp. 1–11. IEEE (2019)
55. Subash, K.M., Kumar, L.P., Vadlamani, S.L., Chatterjee, P., Baysal, O.: Disco: A dataset of discord chat conversations for software engineering research. In: Proceedings of the 19th International Conference on Mining Software Repositories, MSR '22, p. 227–231. Association for Computing Machinery, New York, NY, USA (2022)
56. Tawosi, V., Al-Subaihin, A., Moussa, R., Sarro, F.: A versatile dataset of agile open source software projects. In: Proceedings of the 19th International Conference on Mining Software Repositories, MSR '22, pp. 707–711. Association for Computing Machinery, New York, NY, USA (2022)
57. Uttl, B., White, C.A., Gonzalez, D.W.: Meta-analysis of faculty's teaching effectiveness: Student evaluation of teaching ratings and student learning are not related. Stud. Educ. Eval. **54**, 22–42 (2017) Evaluation of teaching: Challenges and promises
58. Vidoni, M.: Evaluating unit testing practices in r packages. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), pp. 1523–1534 (2021)
59. Vidoni, M.: Self-admitted technical debt in r packages: An exploratory study. p. 11. IEEE, Madrid, Spain (2021)
60. Wan, C., Liu, S., Hoffmann, H., Maire, M., Lu, S.: Are machine learning cloud apis used correctly? In: 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), pp. 125–137 (2021)
61. Warrick, M., Rosenblatt, S.F., Young, J.G., Casari, A., Hébert-Dufresne, L., Bagrow, J.: The ocean mailing list data set: Network analysis spanning mailing lists and code repositories. In: Proceedings of the 19th International Conference on Mining Software Repositories, MSR '22, pp. 338–342. Association for Computing Machinery, New York, NY, USA (2022)
62. Wieringa, R.J.: Design science as nested problem solving. In: V.K. Vaishnavi, S. Purao (eds.) Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology, DESRIST 2009, Philadelphia, Pennsylvania, USA, May 7-8, 2009. ACM (2009)
63. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B.: Experimentation in Software Engineering. Springer (2012)
64. Xiao, W., He, H., Xu, W., Tan, X., Dong, J., Zhou, M.: Recommending good first issues in github oss projects. In: Proceedings of the 44th International Conference on Software Engineering, ICSE '22, pp. 1830–1842. Association for Computing Machinery, New York, NY, USA (2022)
65. Yan, Y., Frey, S., Zhang, A., Filkov, V., Yin, L.: Github oss governance file dataset. In: 2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR), pp. 630–634. IEEE Computer Society, Los Alamitos, CA, USA (2023)
66. Young, J., Casari, A., McLaughlin, K., Trujillo, M.Z., Hébert-Dufresne, L., Bagrow, J.P.: Which contributions count? Analysis of attribution in open source. In: 18th IEEE/ACM International Conference on Mining Software Repositories, MSR 2021, Madrid, Spain, May 17–19, 2021, pp. 242–253. IEEE (2021)

67. Zacchiroli, S.: A large-scale dataset of (open source) license text variants. In: Proceedings of the 19th International Conference on Mining Software Repositories, MSR '22, pp. 757–761. Association for Computing Machinery, New York, NY, USA (2022)
68. Zhou, X., Huang, H., Zhang, H., Huang, X., Shao, D., Zhong, C.: A cross-company ethnographic study on software teams for devops and microservices: Organization, benefits, and issues. In: IEEE/ACM 44th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), pp. 1–10. IEEE Computer Society, Los Alamitos, CA, USA (2022)

Ten Years of Teaching Empirical Software Engineering in the Context of Energy-Efficient Software

Ivano Malavolta, Vincenzo Stoico, and Patricia Lago

Abstract In this chapter, we share our experience in running ten editions of the Green Lab course at Vrije Universiteit Amsterdam, the Netherlands. The course is given in the Software Engineering and Green IT track of the computer science master's program of VU. The course takes place every year over a 2-month period and teaches computer science students the fundamentals of empirical software engineering in the context of energy-efficient software.

The peculiarity of the course is its *research orientation*: at the beginning of the course, the instructor presents a catalog of scientifically relevant goals, and each team of students signs up for one of them and works together for 2 months on their own experiment for achieving the goal. Each team goes over the classic steps of an empirical study, starting from a precise formulation of the goal and research questions to context definition, selection of experimental subjects and objects, definition of experimental variables, experiment execution, data analysis, and reporting.

Over the years, the course became well-known within the software engineering community since it led to several scientific studies that have been published at various scientific conferences and journals. Also, students execute their experiments using *open-source tools*, which are developed and maintained by researchers and other students within the program, thus creating a virtuous community of learners where students exchange ideas, help each other, and learn how to collaboratively contribute to open-source projects in a safe environment.

I. Malavolta (✉) · V. Stoico · P. Lago
Vrije Universiteit Amsterdam, Amsterdam, The Netherlands
e-mail: i.malavolta@vu.nl; v.stoico@vu.nl; p.lago@vu.nl

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2024

D. Mendez et al. (eds.), *Handbook on Teaching Empirical Software Engineering*,
https://doi.org/10.1007/978-3-031-71769-7_8

1 Introduction

Developing energy-efficient software is not an option anymore; it is a matter of survival and moral duty. The Track Clean Energy Progress (TCEP)[1] report of 2023 indicates that data centers and data transmission networks contribute, respectively, to the 1–1.5% of the global energy demand [57]. This corresponds to approximately 240–340 TWh for data centers alone, more than double the amount needed to power the Netherlands in 2020 (i.e., 115.88 TWh). Despite evidence that technological improvements helped mitigate their energy demand in the past years, the request for data center resources and network traffic is increasing. Especially with the expansion and large-scale application of AI-based solutions like generative AI (e.g., to create new contents like text, images, and videos) [23, 57], there is growing evidence that the electricity consumption (and likely the carbon footprint) of software will escalate in the next years [63, 64].

With such high stakes, it is fundamental that practitioners follow an **evidence-based approach** and make informed decisions about the energy efficiency and carbon footprint of their software [38, 44]. In this context, thanks to its roots in insight- and evidence-oriented nature, the importance of empirical software engineering [69] is already prominent among both researchers and practitioners. However, applying empirical methods to assess the energy aspects of software is a difficult endeavor; precisely measuring the energy efficiency of software in a reliable manner requires strong technical skills, and it has proven to be very challenging, time-consuming, and technologically fragmented [17, 21]. As such, strong competencies in empirical methods applied to energy-efficient software are in high demand [50].

This chapter is about our experience in running ten editions of the **Green Lab** course at the Vrije Universiteit (VU) Amsterdam, the Netherlands. The course is given in the Software Engineering and Green IT (SEG) track of the computer science master's program of VU. The course takes place every year over a 2-month period and teaches Computer Science students the fundamentals of empirical software engineering (ESE) in the context of energy-efficient software.

The main peculiarity of the course is its **research orientation**: at the beginning of the course, the instructor presents a catalog of scientifically relevant problems, and each team of students signs up for one of them and works together for 2 months on their empirical study to solve the assigned problem. Each team goes over the classic steps of an empirical study (a la Wohlin *et al.* [69]), starting from the formulation of the goal and research questions to context definition, selection of experimental subjects and objects, definition of experimental variables, experiment execution, data analysis, and reporting. Over the years, the course became well-known within the international software engineering research community since it led to several scientific studies published at various scientific conferences and journals, such as

[1] The Track Clean Energy Progress is a report published by the International Energy Agency (IEA) that evaluates the latest progress regarding the transition to clean energy sources.

EASE, MOBILESoft, IST journal, ICT4S, etc., even winning best and distinguished paper awards.

The course has a **hands-on nature**. Student projects (i) use real-world software products as experimental subjects (e.g., by mining the front end of publicly available Web sites, mining mobile apps, or bringing up widely used open-source platforms) and (ii) involve the execution and measurement of experimental subjects on a dedicated hardware (e.g., smartphones, servers, laptops, IoT devices). The hands-on nature of the course exposes students to recurrent concerns (and their corresponding trade-offs) that software engineering researchers have when designing and conducting controlled experiments on the energy-efficiency of software. Examples of such concerns include the selection of representative objects/subjects, reliability and precision of the energy measurement tools, feasibility of the experiment (and corresponding decisions in terms of experiment design), correctness of the statistical analysis, and representativeness of the obtained results.

Students execute their experiments using **open-source tools** (e.g., Android Runner [35]), which are developed and maintained by the instructors of the course and other students within the program, thus creating a virtuous community of learners where students exchange ideas, help each other, and learn how to collaboratively contribute to open-source projects in a safe environment.

The **intended target audience** of this chapter are instructors willing to include research- and empirical-oriented activities into computer science and software engineering programs. We designed the Green Lab course as a *technology-independent course*, i.e., its learning objectives, structure, assessment procedure, and lectures are exclusively about energy efficiency and ESE research methods, and not bound to any specific technology or application domain. Instructors can access the study guides of the course, its slide sets, links to replication packages of experiments, and links to open-source tools in the **supplementary online material** of this chapter [55].

The remainder of this chapter is organized as follows: Sect. 2 presents the educational context in which the Green Lab course is embedded. Section 3 presents the design of the course, its learning objectives, and main educational components. Section 4 describes the equipment and measurement infrastructure supporting the experiments developed in the context of the Green Lab course. Section 5 presents the open-source tools developed by the Green Lab community. Representative examples of scientific studies emerging from the Green Lab are presented in Sect. 6, while we elaborate on lessons learned for instructors in Sect. 7. We wrap up the chapter and provide our planned future improvements of the course in Sect. 8.

2 Educational Context

As mentioned in Sect. 1, the Green Lab course is part of the SEG track, which provides one of the possible specializations within the 2-year computer science master's program at VU Amsterdam. At the time of creating the track [29], the relation between software engineering and energy was barely understood. Green IT,

instead, was a term emerging yet resonating among people. Hence, we decided to name the track after this term. However, we used it in its broad definition (so as to include software too), i.e., "*the study and practice of designing, manufacturing, using, and disposing of computers, servers, and associated subsystems efficiently and effectively with minimal or no impact on the environment*" [40].

The SEG track entails five so-called *core* courses. Each course counts 6 credits according to the European Credit and Transfer Accumulation System (ECTS), hence counting a total of 30 ECTS. To integrate energy awareness in the track, we have adopted a mix of the *distributed approach* and the *centralized approach* identified by Mann *et al.* [37]. Accordingly, we both revisited pre-existing courses across the whole track (*distributed approach*) and created a few dedicated specific courses *focusing on sustainability while also having sustainability issues addressed across the curriculum* (blended approach) [37]. The Green Lab belongs to the latter, by teaching competencies that *blend* empirical software engineering in the specific context of energy-efficient software).

Further, we adopted a distributed approach for the other core courses, service-oriented design, digital architecture, and fundamentals of adaptive software (by applying software engineering competencies to practical projects and assignments that include Green IT aspects), and a centralized approach for core course software testing (by teaching traditional software engineering competencies that can possibly be used to test software for e.g., detecting energy hotspots [46]).

Timewise, the academic year at VU Amsterdam is organized into two semesters, each including three periods of 2-2-1 month of 8-8-4 weeks, respectively, where the last week is dedicated to exams. From a student perspective, an 8-week period should include two courses that require 50% of the student's time, each, while a 4-week period is dedicated to a single course full time. For instance, the Green Lab course is taught in an 8-week period, in parallel with another course.

Finally, according to the academic level of their learning objectives, courses are categorized as at specialized (400), research-oriented (500), or highly specialized (600) level. For instance, from the academic year 2024/2025 the Green Lab course will be classified as a 500-level course, thanks to the consolidation of energy-efficiency software engineering and measurement as fundamental research competencies.

3 Course Design

This section presents the main educational components of the current edition of the course, including the students' cohort, learning objectives, and teaching team (Sect. 3.1), course contents and structure (Sect. 3.2), and assessment method (Sect. 3.3).

3.1 Students' Cohort, Learning Objectives, and Teaching Team

In terms of **students' cohort**, this course has been designed with computer science master's students in mind. Given the technical nature of the course, the recommended background knowledge of students attending the course includes (i) basic statistical analysis techniques (i.e., descriptive statistics) and most common tests and (ii) basic programming/scripting skills. Both requirements are not formally enforced when students enroll in the course, but the lecturer informs students about them in the first lecture of the course, so as to set the expectations and to allow students to prepare for the rest of the course. Students are not expected to be knowledgeable of ESE research methods, which are part of the learning objectives of the course.

In the last edition of the course (2023/2024 academic year), the course was attended by 91 active students. The course attracts a variety of students coming from several programs and master's tracks. Specifically, within the computer science master's program of VU Amsterdam, students' provenance is composed as follows: 43 students of the Software Engineering and Green IT master's track described in Sect. 2 and 18 students of other tracks within the same computer science master's program (14 in the big data engineering track, 2 in the foundational computing and concurrency track, and 2 in the Internet and Web technology track); a total of 24 students come from international tracks of the computer science master's program, of which 20 come from the Software Engineering for the Green Deal (SE4GD) track and 4 come from the Global Software Engineering European Master (GSEEM) track and 1 in the parallel computing systems track; finally, five students come from other master's programs, including (i) parallel computing systems at VU Amsterdam, (ii) computer systems security at VU Amsterdam, (iii) human computing interaction at Utrecht University, and (iv) computer science at the University of Zurich.

Over the years, students' ages, genders, nationalities, scientific backgrounds, and technical skills became more and more heterogeneous. For the sake of students' privacy, we do not report students' personal demographics. Figure 1 shows the most recurrent technical skills of the students enrolled in the last edition of the course. The heterogeneity of students' technical skills and backgrounds has been taken care of when designing the course and mitigated in the assessment phase of the course by allowing students to work in their projects with the technologies they are more familiar with (see Sect. 3.3).

Upon completion of this course, the **learning objectives** in Table 1 are achieved. In the table, we report also the levels of learning covered by each learning objective according to the revised version of Bloom's taxonomy of learning [4, 7].

The **teaching team** is composed of two lecturers and three teaching assistants. The responsibilities of the five members of the teaching team are:

- Lecturer 1: course coordination, giving all lectures on ESE methods (see Sect. 3.2 for the details), acting as the main contact point for students and colleagues, and assessment of 35% of students' projects



Fig. 1 Technical skills of Green Lab students in 2023/2024

Table 1 Learning objectives of the Green Lab course

ID	Learning objective	Bloom's levels of learning
LO1	To learn the principles of empirical experimentation in the field of software engineering	Remembering-understanding
LO2	To measure and assess the impact of software on energy consumption and environmental footprint	Understanding-applying
LO3	To become familiar with and critically reflect on the research results, trends, and challenges in the field of green software engineering	Understanding-analyzing
LO4	To design and conduct successful experiments on the energy efficiency of software	Applying-analyzing-evaluating-creating
LO5	To synthesize, correctly interpret, and put into context the results of an experiment on the energy efficiency of software	Evaluating

- Lecturer 2: giving the lecture on green software (see Sect. 3.2 for the details), supporting students experiencing technical issues with respect to servers and IoT devices, advising students in choosing and configuring measurement tools, and assessment of 35% of students' projects
- Teaching assistant 1: giving the lab on the Green Lab environment, tools, and measurement tools (lab 1, see Sect. 3.2), supporting students experiencing technical issues with respect to mobile devices and apps, and assessment of 10% of students' projects

- Teaching assistant 2: giving labs on data analysis with R[2] (labs 2, 3, and 4; see Sect. 3.2), and assessment of 10% of students' projects
- Teaching assistant 3: supporting students experiencing general technical issues and assessment of 10% of students' projects

In the 2023/2024 edition of the course, lecturer 1 is an associate professor in software engineering, with experience on ESE applied to energy-efficient software, lecturer 2 is a post-doc researcher specialized in software performance engineering and energy-efficient software and is also responsible for the daily activities within the Green Lab servers (see Sect. 4). The three teaching assistants are master's students; teaching assistant 1 is also the main developer of the open-source tools used in the Green Lab course (see Sect. 5). The structure of the teaching team emerged organically over the years, and it is influenced by (i) the growing number of students in the course,[3] (ii) available resources, and (iii) the specific research interests of involved lecturers (and teaching assistants, to a lesser extent).

3.2 Course Contents and Structure

The course is designed to expose students to the fundamentals of empirical software engineering (learning objective LO1) in the context of energy-efficient software (learning objectives LO2, LO3, LO4, and LO5). As such, the course covers first the basic principles of ESE (e.g., focus on applicability, quantitative vs qualitative research methods); then, given the importance of measurement-based studies in the context of energy-efficient software, it delves into the specifics of the *controlled experiment* method with software subjects and objects (as opposed to experiments with human subjects [28]). The overall organization of the course is presented in Fig. 2.

As anticipated in Sect. 2, the duration of the course is 8 weeks, with lectures and labs being given in the first 7 weeks. In each of the first 7 weeks, there are two 2-hour educational components in which the teaching team gives either a lecture or a lab. The majority of the lectures (depicted in black in Fig. 2) are about ESE research methods; those lectures are designed so that (i) their main contents and principles are generic, and (ii) the lecturer systematically refers to examples, scientific results, cases, and success stories related to energy-efficient software. This makes the course *future-proof* for both students and lecturers: (i) students are able to apply the learned ESE research methods also in contexts different from energy-efficient software (e.g., in their final thesis or during their professional career), and (ii) every year, lecturers update only the energy-specific material (e.g., scientific studies, cases) without

[2] https://www.r-project.org

[3] The first edition of the course had five students. Five years ago, it had 21 students, and in its last edition (2023/2024), it had 91 students.

Week	Day 1	Day 2	Assignments
1	**L1** - Introduction to the course; Example of team project; Intro to Empirical Software Engineering [Lecturer]	**L2** - Experimental Process; GQM [Lecturer]	
2	**LAB1** - Lab environment, tools, and devices (Android/Experiment Runner) [TA 1]	**L3** - How to design and develop green software [Lecturer]	Assignment 1
3	**L4** - Experiment planning; Subjects and variable selection; Measurement theory basics [Lecturer]	**L5** - Experiment Design (basics and advanced) [Lecturer]	
4	**L6** - Data Analysis; Hypothesis Testing [Lecturer]	**LAB2** - R in practice [TA 2]	Assignment 2
5	**L7** - Statistical Tests [Lecturer]	**LAB3** - Statistical tests with R [TA 2]	
6	**L8** - Data Visualization [Lecturer]	**LAB4** - Data visualization in R with ggplot2 [TA 2]	
7	**L9** - Experiment Reporting; Validity evaluation [Lecturer]	*Guest lecture*	
8	FINALIZE PROJECT		Assignment 3

Fig. 2 Schedule of the course in 2023/2024

needing to completely restructure the course. The course includes also a special lecture about how to design and develop green software (L3, depicted in blue in Fig. 2); this lecture emerged from the students' feedback we received in the previous years of the course, where students were signaling the fact that the course was "too meta" on how to measure the energy-efficiency of software, instead of providing concrete advice on how to make software more energy efficient. During lab sessions (depicted in orange in Fig. 2), students are assisted for technical operation of the lab equipment as regards measurement and tools (see Sects. 4 and 5 for details). In the labs, students also receive the required training for data analysis and visualization using R and RStudio.[4] In order to provide students with the wider perspective on ESE methods and/or energy-efficient software, the last educational component of the course is a guest lecture. The guest lectures we had in the years are *Leveraging Implementation Diversity to Improve Energy Efficiency* (2023, Fernando Castor, University of Twente), *Sustainability and Cloud* (2022, Thiago de Faria, Amazon Web Services), and *Energy-Efficient Robotic Software* (2021, Michel Albonico, Federal University of Technology Paraná).

Throughout the course, students work in teams to perform experiments on energy-efficient software. Students carry out all the phases of a controlled experiment, from experiment design to execution, data analysis, and reporting. Students are provided with examples of experiments coming from state-of-the-art literature, but they will have to choose by themselves the experimental subjects and hypotheses to test. Further details about the project are given in Sect. 3.3.

At the end of every lecture, the lecturer provides a series of (mandatory and optional) readings. The **syllabus** of the course is defined as follows:

- Textbook: The book "Experimentation in Software Engineering" by Wohlin *et al.* [69] is the main textbook of the course, and the majority of the contents of given lectures follow the same structure of the book.
- Complementary mandatory readings: the textbook is complemented by (i) Chapters 6 and 8 of the book "Guide to advanced Empirical Software Engineering" [53] about statistical methods, measurement theory, and experiment reporting and (ii) Chapters 5 and 16 of the book "Basics of software engineering experimentation" [27] about designing and reporting empirical studies.
- Support material: additional sources are provided to the students as optional material to be used as a manual/documentation in case students need to go deep on specific aspects of their projects. Specifically:
  - Material on ESE research methods: a selection of scientific studies and guidelines for conducting empirical studies. For example, such studies are about statistical analysis in ESE [16, 19], data analysis in ESE [60], and experiment design in ESE [43].
  - Material on energy efficiency: a selection of studies providing either the bigger picture about energy-efficient software or methodological guidelines. For

[4] https://posit.co/download/rstudio-desktop

example, such studies are about Green IT and green software in general [62], practitioners' perspectives on green software [38, 44], and guidelines for energy measurement and its related issues [5, 45].
  - Material on statistics and data visualization: a selection of studies and guidelines about statistics and data visualization in general, e.g., [52] and [51].

- Articles on well-conducted experiments (e.g., [12, 31, 49]): a selection of well-designed and reported empirical studies; the lecturer uses such studies in class as success stories and as triggers for critical reflection. This part of the syllabus also includes scientific studies emerging from previous editions of the Green Lab (see Sect. 6); those studies help in setting the standard for the expected quality of students' projects.
- Programming books: books about R [68], Python [33], and Bash programming;[5] these are the programming languages that are very likely used across all projects. Students are invited to use those books to refresh their technical skills and solve specific issues they might have during the project.
- Videos: a selection of YouTube videos recorded by researchers for scientific conferences like ICSE and MSR.[6]

The course counts as 6 ECTS credits. Since each ECTS credit is equivalent to 28 hours of study, the total **expected effort of each student** is 168 hours. To set students' expectations right, in the first lecture of the course, we advise the following breakdown of activities: (i) in each of the first 7 weeks, the student invests 4 hours for the two educational components, 8 hours of study, and 8 hours of work on the project; (ii) the remaining 28 hours are used for finalizing the project in the final week of the course.

3.3 Assessment

Teams of five students conduct a concrete research project throughout the whole course. Each part of the research project is discussed during the lectures, started during the labs, and completed as homework, so as to keep students on track within the course schedule.

The goal of the research project is to plan, design, conduct, and report a scientific experiment in the context of the energy efficiency of various types of software (e.g., mobile apps, software libraries, microservices). Each team works on a specific topic in the context of energy-efficient software. In this way, students put into practice the skills and techniques that they have learned during the lectures/labs and develop

[5] https://www.systemcodegeeks.com/minibook/linux-bash-programming-cookbook

[6] Using videos also gives students a "human" touch to the scientific studies they read during the course, reminding them that researchers are also human beings and sometimes are just a few years older than them.

their practical insights by applying them to real software. In the first lecture of the course, the lecturer provides a list of possible topics; then each team indicates (i) its members, (ii) the technical skills and educational background of each member, and (iii) a set of scores indicating the preferences about each proposed topic. Examples of potentially assignable topics are as follows: (i) What is the energy efficiency of AI-generated algorithms with respect to human-created algorithms? (ii) How do configuration settings of the Zipkin monitoring tool impact the energy consumption of a Docker-based system? (iii) How does WebGPU compare against WebGL in terms of the energy efficiency of Web apps? The topics are assigned to teams based on preferences, technical skills, and knowledge indicated by each team after the first lecture of the course. Then, each team is responsible for independently carrying on the experiment on the assigned topic. When possible, the lecturer provides relevant datasets, scripts, and other material to the team in order to smoothen the execution of their experiment. Such information is typically provided during the first lecture of the course (contextually to the description of the topics for students' projects), during the first lab, during the in-person discussions with students before or after lectures/labs, or contextually to the grading (i.e., as part of the feedback given to students on their assignments); moreover, in case a new relevant dataset, tool, or scientific study is published, lecturers make an announcement on the Web platform of the course about it so that students can evaluate whether the new material can be used in their project.

As shown in Fig. 2, each team project is composed of three assignments. Each assignment deals with a specific increment of the same research project. The final assignment is a fully fledged scientific study, including all phases of a classical ESE study reporting on a measurement-based controlled experiment. The sequence of assignments is designed in such a way that their difficulty and required effort grow together with students' deeper understanding of ESE principles, the nuances of their research project, and their experience in terms of energy measurement. We believe that such incremental design of the research project is important to create a safe environment for students to explore and study at the beginning of the course and then let the project grow together with them during the course.

All assignments include a **written report** describing all the information related to a specific phase of the project. A complete template for Green Lab reports is publicly available on Overleaf.[7] Written reports contain also a link to a *time log*, where students record their time; the time log is used only in case of disputes among team members or in case of strong suspicions of fraud.

Below we describe the main components of each of the three assignments.

- **Assignment 1: context and experiment definition** (20% of the final grade). This part is composed of three sub-components:
  - *Technical/societal context*: students describe (i) the domain (e.g., mobile apps and their market) and the technologies relevant to the experiment; (ii)

[7] https://www.overleaf.com/latex/templates/green-lab-report-template/cpchhrgcrnrr

the main motivation behind the experiment, including examples, apps/tools screenshots, snippets of source code, etc.; (iii) a brief overview of the direction of the experiment; and (iii) what the target audience (e.g., software developers) will learn from the results of the experiment.
  - *Scientific context*: students position their research by elaborating on five to eight related studies that are similar in terms of goal or methodology, with a clear characterization of the novelty of their own experiment with respect to them.
  - *Experiment definition*: students use the Goal-Question-Metric (GQM) approach [6, 8] to formulate the goal, research questions, and high-level metrics (e.g., power, execution time, energy, CPU usage) of their experiment. Using the GQM approach helps students (and instructors) get a complete and concise overview of the main direction of the experiment.

- **Assignment 2: experiment planning and measurement infrastructure setup** (30% of the final grade). This part is composed of two sub-components:
  - *Experiment planning*: students make a detailed plan about the experiment in terms of (i) subjects' selection/mining, (ii) experimental variables (i.e., dependent and independent variables), (iii) formal definition of experimental hypotheses, (iv) experiment design (i.e., main, co-, fixed, and blocking factors and their organization), and (v) data analysis procedure. In this phase, students also provide an estimation of the total duration of the experiment considering the total number of subjects, trials, runs, and estimated execution time for each run; over the years, this proved to be a valuable tool for facilitating students' reasoning on the feasibility of their experiment and take countermeasures, e.g., by reducing the number of subjects or moving from a full-factorial design to an incomplete one. In general, we advise students to design their experiment in such a way that its execution takes less than 40 hours.
  - Measurement infrastructure setup: students present the technical aspects for executing the experiment, which tools they are going to use, which devices/servers, and the software/hardware infrastructure they are setting up for the experiment.

- **Assignment 3: experiment execution, data analysis, and reflection** (50% of the final grade). This part is composed of five sub-components:
  - *Experiment execution*: students execute the experiment and collect measures as planned in the second assignment (after having addressed the instructor's feedback). The output of this phase is the raw data collected from the executed experimental runs.
  - *Data analysis and results reporting*: students check the correctness of the collected measures, compute descriptive statistics, perform the statistical analysis for hypotheses testing (with effect size estimation), and synthesize the obtained results in their report providing suitable plots and tables to illustrate the main points of attention.

- *Threats to validity*: students reflect and report the threats to the validity of the experiment according to the Cook and Campbell classification [14].
- *Reflection*: students report the main implications and interpretations of the obtained results (possibly grouped by research question). In this phase, since they spent weeks on the technical aspects of the experiment, students tend to have a narrow perspective and focus very much on the low-level, technical aspects of the experiment. So, here, students are asked to address directly the target audience of their experiment (i.e., who will benefit from the results of the experiment), as described in the previous assignments; this helps students in getting a wider perspective about their experiment and the general usefulness of their results.
- *Report finalization*: students wrap up the project by doing a final complete pass on the report and writing the abstract and conclusion section.

Assignment 3 includes also a link to a GitHub repository containing the complete **replication package** of the experiment. The replication package must contain (i) source code of the scripts developed for running the experiment, (ii) source code of any software developed for building used dataset, (iii) raw data resulting from the execution of the experiment, (iv) R scripts for data analysis, and (v) any other relevant material for replicating the experiment. A template for replication packages is provided at the beginning of the course in the form of a GitHub repository.[8] Assignment 3 includes also a link to a **video** where the team presents the main aspects of the experiment in a complete manner (from the motivation and context, design, and execution to the results, discussion, etc.). Presentations are prepared as groupwork where each student presents individually a part of the experiment. Presentations are assessed as either `pass` or `fail`.

When grading each assignment, the teaching team provides (i) a quantitative assessment of each part of the assignment according to a shared rubric (available in the supplementary material of this chapter) and, more importantly, (ii) detailed feedback on how to improve the experiment. Intermediate assignments are evaluated and are part of the final assessment of the whole team project (assignment 3). Assignment 3 is a coherent integration of the previous assignments. When working on assignments 2 and 3, student teams are requested to address the feedback provided by the teaching team in the preceding assignment. Over the years, feedback on intermediate assignments proved extremely useful since it allowed the teaching team to (i) ensure the feasibility of the project (in some cases, students tended toward overpromising), (ii) ensure that the difficulty of assigned projects is similar among all projects within the cohort of students, and (iii) steer projects toward scientifically challenging and novel research directions.

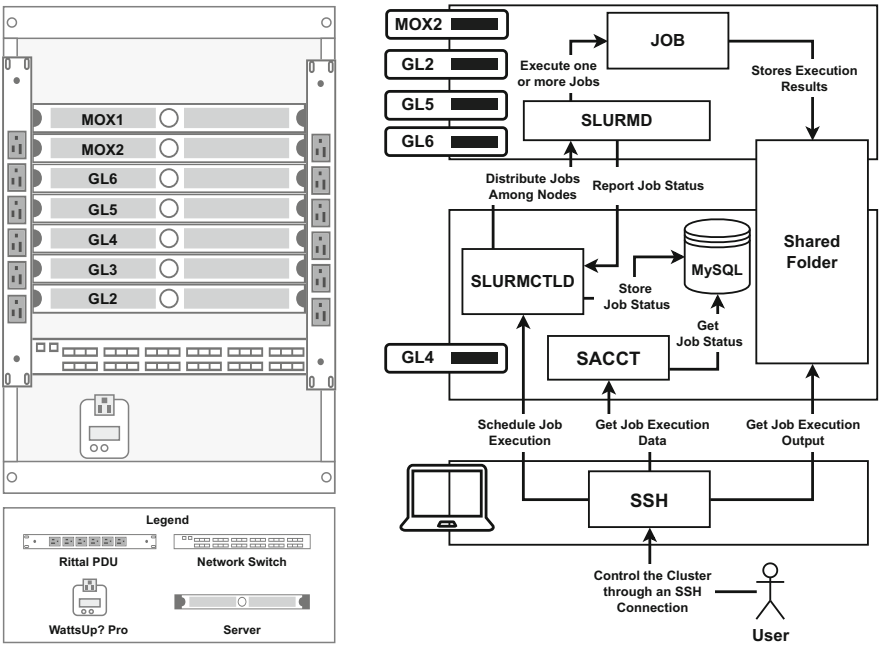
[8] https://github.com/S2-group/template-replication-package

4 Equipment and Measurement Infrastructure

Green Lab empowers researchers and students to explore diverse software-related domains. So far, the Green Lab has been used for experiments on distributed, mobile, robotics, artificial intelligence (AI), virtual reality (VR), and embedded software applications. The Green Lab includes a *cluster* of seven servers with different specifications. Table 2 presents an overview of the hardware specifications of each server, along with the operating system currently in use, while Fig. 3a

Table 2 Green lab servers specification

ID	HD	RAM	CPU (Intel Xeon)	Operating system
MOX1	36 Tb	196 Gb	Silver 4112@2.60 GHz	Debian 9
MOX2	36 Tb	384 Gb	Silver 4208@2.10 GHz	Debian 11
GL6	36 Tb	384 Gb	Silver 4208@2.10 GHz	Ubuntu 20.04
GL5	36 Tb	384 Gb	Silver 4208@2.10 GHz	Ubuntu 22.04
GL4	1 Tb	16 Gb	E5335@2.00 GHz	Ubuntu 22.04
GL2	1 Tb	32 Gb	E3-1231@3.40 GHz	Ubuntu 22.04
GL3	126 Gb	8 Gb	E5345@2.33 GHz	Ubuntu 22.04



(a) (b)

Fig. 3 The Green Lab physical infrastructure. (**a**) Green Lab physical servers. (**b**) Green Lab – HPC configuration

represents the cluster rack. MOX1, MOX2, GL5, and GL6 are the most powerful servers in the cluster. The cluster supports different ways of executing the software and managing the resources of the servers. Indeed, it is configured to run virtual machines and containers, deploy an application on a single node, as well as distribute and parallelize software across the cluster. The cluster enables users to isolate software applications on a testbed and control the factors influencing their energy consumption and performance. For example, users might be interested in generating different synthetic workloads and injecting them into the application running in our cluster. As a result, it is possible to discover potential energy/performance hotspots and understand how the application reacts under stress.

MOX1 runs Proxmox [22], an open-source server virtualization software, and, therefore, is dedicated to *virtualization*. Proxmox enables students and researchers to create virtual machines on-demand for testing software systems, executing data analysis, and monitoring other servers. Proxmox supports two virtualization technologies: Kernel-based Virtual Machines and Linux Containers. Moreover, it allows dynamic scaling of resources assigned to each virtual machine, enabling the deployment of different-size workloads at runtime. Proxmox offers a Web-based interface for creating, removing, and allocating resources to virtual machines remotely. The cluster also supports *containerization*. Indeed, container-based applications, such as SockShop [67] and Train Ticket Booking System (TTBS) [20], can be deployed either on a single machine and across the servers.

The cluster is configured to offer all the computational resources of the servers and, thus, supports *High-Performance Computing* (HPC) applications that need to execute compute-intensive workloads, such as simulation, machine learning models, and data analytics. To do this, we use SLURM [70], an open-source workload manager that handles the distribution and parallelization of computation across the nodes of the cluster. SLURM allows users to create and distribute computation and monitor the energy consumption and performance of the jobs and nodes within the cluster. Figure 3b describes a typical execution of a job on our cluster. Figure 3b shows the current configuration of our HPC cluster, which includes a head node, i.e., GL4, and four compute nodes, namely, GL5, GL6, GL2, and MOX2. The HPC cluster is arranged as a tree, which means that the head node creates and distributes jobs on the compute nodes. A user can set the jobs to be executed, allocate resources to the jobs, retrieve statistics about job execution, and retrieve the output resulting from job execution. These commands are handled by the `slurmctld` daemon that runs on the head node. This daemon communicates with the `slurmd` daemons that run on each compute node. Each `slurmd` daemon is responsible to track job execution and report information to the head node. These statistics include the status of the node, information about job status as well as job performance, and energy consumption. The head node records the information about job execution in a MySQL database. The interactions between the database and the `slurmctld` daemon are managed by the `slurmdbd` daemon that we omitted in Fig. 3b for simplicity. The output resulting from jobs execution is stored into a shared folder that is accessible by all the nodes of the cluster. A user can collect the information

stored in the database using the `sacct` command of SLURM and the data contained in the shared folder through SSH.

The Green Lab includes about 20 smartphones and 3 tablets for executing experiments on *mobile* applications, including several generations of Google Pixel phones, Samsung J7 phones, etc.. In addition, the Green Lab has 32 Raspberry Pis (of which 2 are Pi 5 Model B, 16 Pi 4 Model B, and 14 Pi 3 Model B) and 8 Arduino Nano to test embedded and IoT software applications. For experiments involving *AI models*, in addition to using the cluster, the lab has also 3 Jetson Nano development kits. For *robotics* systems, the Green Lab has five TurtleBot3 Burger robots (equipped with a camera module and current sensors) and a four-degree-of-freedom robot arm (UARM Swift Pro, UFactory). Additional equipment includes *wearables*, such as Samsung Gear S3 and *virtual reality* visors, i.e., a Google Daydream, and a *drone*, namely, a DJI Tello.

4.1 Measurement Infrastructure

Green Lab users can track energy consumption and performance of the software running in the lab using both software energy profilers and hardware power meters. The cluster is equipped with two Rittal Power Distribution Unit (PDU) [48] and a Watts Up Pro [66] that profile the power requested by each node. A Rittal PDU is a power strip that provides reliable power distribution with energy monitoring and measurement for all devices connected to it. A Watts Up Pro, on the other hand, is a plug power meter that is placed between the device to profile and its power source. Both power meters provide an API to get the measurements. In addition, the Rittal PDU provides a Web interface to configure the power strip and monitor the power required by the servers in real time. At the current state, the two Rittal PDU supply and monitor power of all the nodes in the cluster. We use the Watts Up Pro to profile the power consumption of a node and also of other embedded systems, such as smartphones and Raspberry Pis. The Green Lab provides three Watts Up Pro, including the one connected to the cluster. The Green Lab also features three Monsoon Power Monitor [39], which are high-frequency power meters. We mainly use them for experiments involving mobile applications, but they can also be used to profile other embedded systems. A Monsoon can be plugged into a device through a USB port, a Main Channel consisting of a positive and a negative terminal, and a BNC connector referred to as Auxiliary port in the Monsoon documentation. We use the USB port to profile USB-powered devices and the Main Channel for tracking the battery of our smartphones. The Main Channel can be connected to all devices exposing a positive and negative terminal, such as the pins of a Raspberry Pi. The power consumed by the Raspberry Pis and the Arduino Nano can be monitored also using an INA219 sensor. The INA219 [1] is a current sensor that can be soldered on a board or connected to the board through a breadboard. The Green Lab comprises five INA219 at the moment.

The abovementioned physical power meters record the energy consumption of the whole device while running a software application. A handy solution for fine-grained energy consumption measurements are software energy profilers [15]. Software energy profilers avoid direct interaction with the hardware platform and allow more fine-grained measurements on the software, for example, at the process or code block level. Among the most used software profilers, there are `powerstat` [13], `perf` [32], and `Intel Power Gadget` [26]. All the mentioned tools exploit the Running Average Power Limit (RAPL) [25] interface provided by Intel CPUs. At the moment, RAPL is used by most software energy profilers. RAPL estimates the energy consumed by a device according to a power model, which is proven to be accurate on certain architectures, and stores the estimate in a set of registers called model-specific registers (MSRs). SLURM, which we use for the HPC version of Green Lab, also uses RAPL to derive the energy consumed by the jobs executed in the cluster. However, not all nodes in our cluster support RAPL. The older ones, i.e., GL3 and GL4, do not support RAPL. For this reason, GL4 is used only for dispatching jobs and recording the measurements in the HPC configuration.

4.2 Guidelines for Proper Use of Equipment

It is crucial to provide proper instructions to students on how to use the tools and equipment provided for their projects; major misuses of the tools and equipment can affect the completion of their projects or can even lead to physical hardware damage. The definition of the experiment design during Assignment 2 (see Sect. 3.3) is followed by a phase in which the scientific assistants of the S2 Group interact directly with each group to provide them with support material and instruct them on how to use it. The groups are provided by the scientific assistants with credentials to access a limited set of resources, namely, CPU cores, RAM, disk, and network bandwidth, on the servers according to the requirements of the experiment, as well as instructions on accessing them, usually through SSH. Scientific assistants provide step-by-step guides for setting up embedded devices like Raspberry Pis, virtual reality headsets, and educational robots, as well as instructions for maintenance. For example, we instruct students to always check and set the voltage provided by the power source (e.g., the Monsoon power meter) to values that are not greater than those supported by the device to be powered (e.g., 5V for the Raspberry Pi). The same applies to the physical power meters, i.e., the Watts Up Power Monitor and the Rittal PDU. In addition, we provide students with material that could be helpful in better understanding the usage of the devices, such as links to the manuals of the devices and online user discussions. The Green Lab course includes a lab, mentioned as LAB1 in Sect. 3.2, which covers the use of software tools to manage experiments, measure performance, and track energy consumption. During the lab, the instructor explains how to set up the tools and provides a live demonstration of how to use them for experimentation. The scientific assistants and lecturers are available to support the students with technical problems and questions about the

usage of measurement tools during the whole course. This is usually accomplished through open discussions on the Web platform of the course.

5 Open-Source Tools and Community of Learners

In the first three editions of the course, the development of a proper measurement pipeline for executing the experiment was an additional task for the students. However, we noticed that for projects that were focusing on the same technologies (e.g., projects on the energy consumption of PostgreSQL databases or the front-end of mobile Web apps), students tended to "reinvent the wheel" in their scripts in each of their projects. Examples of redundant scripts include tasks such as defining the order of execution for experiment trials, shuffling experiment runs based on the considered subjects and treatments of the main factors, programmatically launching mobile apps, checking the status of smartphones, configurating energy profilers, etc. Even though those tasks were relatively exciting for students, they were very repetitive and error-prone; we saw an opportunity to save students' (and researchers') time here, and we decided to move from ad hoc development of measurement pipelines to the *configuration* of measurement pipelines via *dedicated tools*.

So, in the months before the 2017/2018 edition of the course, we decided to work on a first tool for the automatic execution of measurement-based experiments. To keep the scope of the tool under control, we decided to focus on experiments targeting mobile apps (both Web and native ones) running on Android devices (both smartphones and tablets). This tool is called **Android Runner**[9] [35]. Android Runner is implemented as a set of Python modules, and as such, it can run on any machine able to run Python code (e.g., laptops, Raspberry Pis, servers). Given a JSON-based configuration of the experiment, Android Runner is in charge of performing diagnostic checks, bringing up (energy) profilers, executing each run of the experiment according to a predetermined plan, collecting and aggregating measures from the profilers, etc. Android Runner communicates with the measured apps via Android Debug Bridge (ADB[10]), the official Android command-line tool for interacting with Android devices. Android Runner is independent of the specific energy profilers (which are implemented as external plugins), allowing users to straightforwardly collect runtime measures via already-existing hardware/software profilers or by integrating their own third-party profiler; Android Runner allows users to use multiple profilers within a single experiment, e.g., for collecting performance and energy measures at the same time (as we did in [41]). Moreover, Android Runner allows users to include their business logic as external Python scripts at specific points within the experiment execution (e.g., before the beginning

[9] https://github.com/S2-group/android-runner

[10] https://developer.android.com/tools/adb

of the experiment, before or after each run, etc.); over the years, external scripts proved to be useful in many situations, e.g., for setting up a local Web proxy recording the traffic generated by the apps, for instrumenting the subjects of the experiment on the fly, for cleaning up the environment between runs, etc.

Over the years, and thanks to the lessons learned while working on Android Runner, we decided to develop other two tools for executing measurement-based experiments: (i) Robot Runner, which is dedicated to robotic systems, and (ii) Experiment Runner, which is a generalization of the previous *-Runner tools, allowing students (and researchers in general) to configure measurement-based experiments in a technology-independent manner.

Robot Runner[11] [56] follows the same principles as Android Runner, with the following main differences: (i) it communicates with experiment subjects via Robot Operating System (ROS[12]) messages instead of Android ADB, (ii) the internal diagnostics checks are specific to ROS-based system, (iii) it is completely independent of the type and number of used subjects (either simulated or real), whereas Android Runner assumes to be interacting with a physical and always-available device, and (iv) the experiment configuration is defined in Python instead of JSON. About the latter point, the rationale for having a Python-based representation of the experiment (instead of JSON) is to allow users to flexibly define the configuration and the business logic of their experiments in a *self-contained manner* [56]. This is aligned with the ROS ecosystem, where mission launch files for the robotic system can be defined in Python.[13]

Experiment Runner[14] is our latest tool, and it embodies several of the lessons learned from both Android and Robot Runner. Specifically, it allows users to (i) programmatically define experiments in terms of a so-called *run table* containing experimental factors and their related treatments, subjects and objects, and their combination into trials; (ii) start, stop, pause, and resume experiments; (iii) run the experiment in automatic mode (without any interruptions) or semi-automatic mode (where the experiment waits for a user's action before at the end of a run before proceeding with the next one—this is useful in cases where a manual action is needed in between runs); (iv) intuitively keep track of the progress of the experiment; and (v) define custom Python callbacks that are invoked when specific events are raised during the execution of the experiment (similarly to Android and Robot Runner). During the execution of the experiment, the run table defined by the user is populated dynamically with the measures collected at each run, without requiring the user to manually aggregate them at the end of the experiment. Similar to Android and Robot Runner, measures are collected via external profilers.

At the time of writing, the abovementioned tools have been used to execute a total of 68 experiments within the Green Lab course and many others outside the

[11] https://github.com/S2-group/robot-runner

[12] https://www.ros.org

[13] https://docs.ros.org/en/iron/Tutorials/Intermediate/Launch/Creating-Launch-Files.html

[14] https://github.com/S2-group/experiment-runner

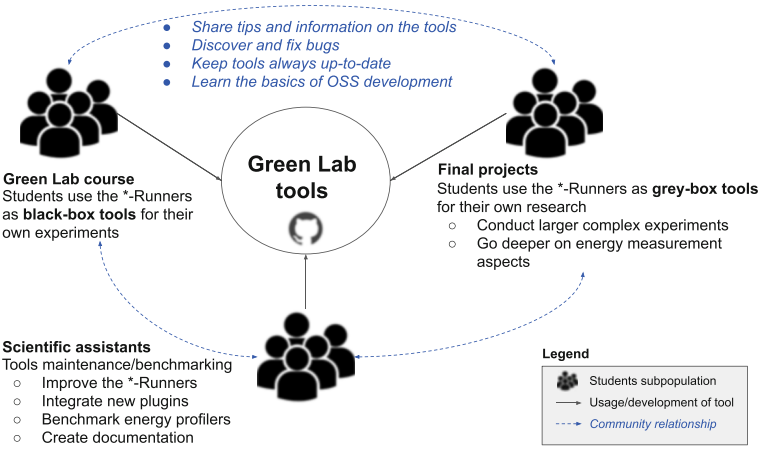


Fig. 4 Green Lab community of learners

scope of the course (e.g., in experiments not linked to education or in students' final projects). As shown in Fig. 4, the three abovementioned tools are the center of a large **community of learners** composed of students attending the Green Lab course, students carrying out their final projects (both bachelor's and master's), researchers of the S2 research group, and students acting as scientific assistants within the S2 group. Specifically, Green Lab students use the "Runners" as black-box tools for executing their own experiments for completing the project of the course; they are not required to know the internal details of the orchestration logic of the tools, but in almost all cases, they integrate the tools with their own custom logic via external Python scripts. Students carrying out the final projects use the Runners for executing their own experiments, which tend to be more complex and large than those of the Green Lab course. Also, they might go deeper on selected aspects of energy measurement, e.g., by benchmarking new tools and profilers, benchmarking already-existing energy meters/profilers against each other, etc. Scientific assistants are generally master's students enrolled within the S2 research group all year long to support the development of the Runners; their responsibilities include improvement of the tools, bug fixing, integration of new plugins and their benchmarking, and documenting the tools via videos, tutorials, examples, guides, and readmes. Together with S2 researchers, these three sub-populations of students form a community of learners where individuals share tips and (less-documented) information about the tools, discover and fix bugs in the tools, and keep the tools up to date. The mechanisms we use for synchronizing all those activities are exactly the same as those of *collaborative software development*.[15] The source code of all Runners is publicly available in their GitHub repositories; learners fork those

[15] https://www.linuxfoundation.org/blog/getting-started-with-open-source-development

repositories, develop and test their fixes or new plugins in their own forks, submit their own contributions to the upstream GitHub repository via pull requests, and receive feedback from S2 researchers and other learners and (if needed) make changes accordingly. The community is growing more and more—as of today, it counts more than 300 participants in total (past and present)—and we believe it is the ideal setup for training students and young researchers to participate in collaborative software development, a skill that is deemed as a must-have in the job market for software engineers.

6 Success Stories

This section presents examples of recent relevant scientific studies that have been designed, conducted, and reported in the context of the Green Lab. All studies have been positively received by the software engineering research community and tended to receive primarily positive reviews by our peers.

Before describing the success stories, it is important to elaborate on how we organize the transition from an education-oriented project developed by five students to a scientific publication able to compete in the software engineering research landscape. This transition is carried out in four phases: (i) projects supervision, (ii) projects selection, (iii) authorship discussion, and (iv) writing and submission.

Projects Supervision This is a long-term phase starting at the beginning of the course and ending with the final submission of students' projects. As already discussed, each project is split into three separate assignments, and instructors provide detailed feedback about each of those assignments (the mindset is to provide a review similar to the ones done in top software engineering conferences). Via those three assessment moments, instructors are able to (i) steer the projects toward research excellence (e.g., avoiding known methodological pitfalls or choosing reliable energy measurement tools) and (ii) ensure that the projects remain on track and relevant to the software engineering community.

Projects Selection The instructors select a subset of students' projects having the highest potential in terms of scientific value. This selection is carried out by the whole team of instructors, and it is based on their overall experience in conducting controlled experiments; in this context, instructors identify and discuss the top projects in terms of (i) robustness of the technical aspects of the experiment, (ii) solidity of the design of the experiment, (iii) replicability of the experiment, (iv) potential scientific relevance and/or practical impact of the experiment, and (v) quality of the reporting of the experiment. The number of projects passing this selection phase range from two to nine projects per edition of the course. The selection phase allows instructors to have a very good starting point for the scientific publication, considering also that the team of students invested more than 800 hours in studying the material and carrying out the project (the expected effort of each student is 168 hours in total for the whole course; see Sect. 3.2).

Authorship Discussion The list of authors of the scientific publication is composed of students (always first in the author list) and one or more members of the teaching team (always last in the author list). Firstly, all students who worked on the project are asked to participate to the scientific publication, clarifying that the additional effort from their side is generally very minimal. Then, those students who accepted to participate in the scientific publication are included in the author list, and the others who did not[16] are included in the acknowledgment section of the paper. Then, one or more members of the teaching team (i.e., the lecturers and all teaching assistants) are associated with the scientific publication. As previously discussed, in the first lecture of the course, the lecturer presents a list of possible topics to students. Such list of topics is created and discussed collaboratively by the whole teaching team; each possible topic is associated with the member(s) of the teaching team who initially proposed it; they act as the *owner* of the scientific publication and will drive the next phase.

Writing and Submission Students are asked to address all feedback provided in the context of their final assignment. Then, the owner(s) of the scientific publication takes care of (i) checking the final revision of the study provided by the student, (ii) rewriting parts that are generally more customary for a scientific publication (e.g., expanding the introduction with a summary of the main results and contributions, revising the related work section, etc.), (iii) expanding and elaborating more in details the main implication of the study (e.g., in the Discussion section), and (iv) forking the initial replication package of the study produced by the students, refining it (when needed), and anonymizing it (in case of double-blind submissions). Students remain available during all the previously listed activities in case clarifications are needed. Finally, the scientific publication is revised by all authors and submitted by the owner.

6.1 Scientific Studies on Web Technologies

6.1.1 Impact of Service Workers on the Energy Efficiency of Web Apps [36]

This was the first scientific study emerging directly from a Green Lab project, and it was awarded with the Distinguished Paper Award at MOBILESoft 2017 (co-located with ICSE). The lessons learned and the various interactions with the students during this study led to the creation of Android Runner [35].

Service workers[17] are a W3C standard providing APIs to allow Web developers to programmatically preload assets required by a Web app, cache data received from

[16] This is a very rare case, and it is primarily due to students having already a full schedule for the remainder of their academic year.

[17] https://www.w3.org/TR/service-workers

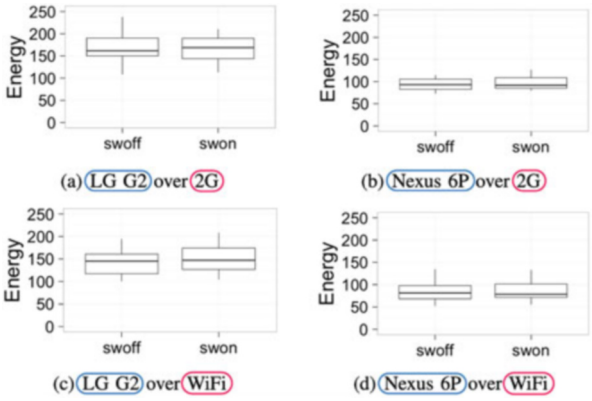


Fig. 5 Energy consumption in Joules of the seven Web apps with (swon) and without service workers (swoff) on different mobile devices (LG G2, Nexus 6P) and under different simulated network conditions (2G, Wi-Fi) (Adapted from [36])

servers, subscribe to and receive push notifications, etc. The main motivation of this study is that service workers were advertised by several technology players as performance boosters, network savers, and providers of better user experience; however, although they are additional codes to be downloaded, parsed, and run by the browser, to this day, nobody investigated the potential overhead in terms of energy consumption. As shown in Fig. 5, this experiment had two main factors: the use of service workers and the type of (simulated) network available (2G and Wi-Fi). The subjects of this experiment are seven third-party and publicly available Web apps, e.g., Washington Post, Ali Express, and Wikipedia. We run the subjects on Google Chrome via two Android devices (LG G2 and Nexus 6P); the Android devices were used as blocking factor. The dependent variable of the study is the energy consumption of the Android devices, measured via the Trepn Power Profiler. The experiment did not provide statistically significant evidence about the impact of service workers on energy consumption, regardless of the network conditions; no interaction was detected between the two main factors of the study.

This study is also relevant in the context of scientific integrity; indeed, it is a good example of the value of non-conclusive studies in terms of statistical significance, allowing students to understand the value of a scientific study independently of reaching statistical evidence.

6.1.2 Correlation Between Performance Scores and Energy Consumption of Mobile Web Apps [10]

This study has been carried out in collaboration with Greenspector, a French company specializing in environmentally sustainable digital transformation of organizations. Greenspector contributed to the study by allowing our students to collect measures via their software-based energy profiler for Android devices.

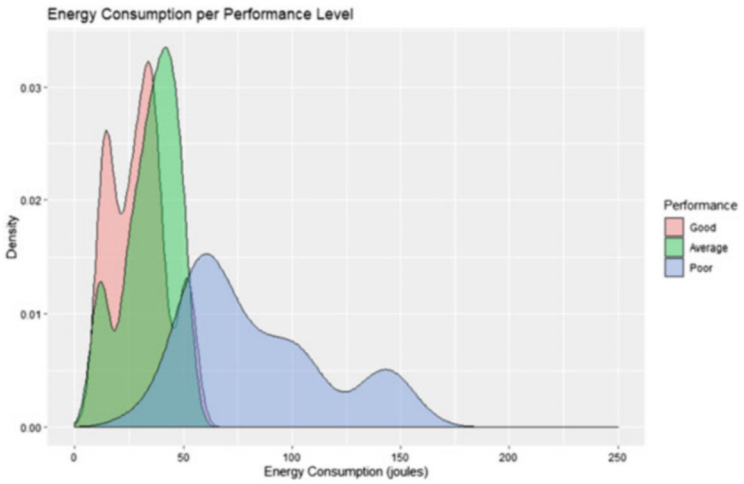


Fig. 6 Density of energy consumption among three performance levels (From [10])

The study is based on the observation that Web developers can use several tools for collecting performance metrics of mobile Web apps (e.g., Google Lighthouse[18]), but similar ready-to-use tools are (still today) not available for energy metrics. The goal of this study is to investigate whether the metrics produced by Google Lighthouse can be used as a proxy for energy consumption. To answer this question, we conducted an experiment where 21 real mobile Web apps (e.g., `apple.com`, `theguardian.com`, etc.) were analyzed in terms of their performance level (via Google Lighthouse) and their energy consumption. After having collected the measurement data for 525 runs (21 Web apps, 25 repetitions each), we statistically analyzed the correlation between the obtained performance metrics and energy consumption and carry out an effect size estimation. The results of the study provided empirical evidence about a statistically significant negative correlation between performance levels and the energy consumption of mobile Web apps, with medium to large effect sizes (see Fig. 6). In practical terms, this means that if a Web app has a good performance score on mobile devices, then developers can use such a score as a low-cost alternative for preliminary insights about its energy consumption.

This study attracted the attention of Web developers, and it is used as one of the support resources of the current draft of the Web Sustainability Guidelines,[19] a specification providing recommendations for making Web sites and products more sustainable published by the W3C Sustainable Web Design Community Group.

[18] https://developers.google.com/web/tools/lighthouse

[19] https://w3c.github.io/sustyweb

6.1.3 Performance and Energy Costs of Ads and Analytics in Mobile Web Apps [41]

This study is published in the Information and Software Technology journal, and it is the first journal publication emerging from a Green Lab project.

This study targets ads and analytics in Web apps running on mobile devices. The premise of the study is that ads and analytics have inherent costs due to them being additional software modules running in a Web app and making additional network requests. Subsequently, more computing resources are used, potentially having an impact on the energy consumption and the performance of Web apps. This study aims to analyze the performance and energy overhead of ads and analytics on mobile Web apps. The results of this research are intended to provide developers, browser vendors, and researchers with insights into the energy consumption and performance drain caused by running such software on consumer mobile devices. Students sampled nine popular Web apps containing both ads and analytics from the Tranco list [30], a stable research-oriented list of the top one million most popular Web applications. For each Web app, students obtained three versions: the full original Web app, a version without ads, and a version without analytics. Then, each version of each Web app is loaded multiple times on an Android device via two different browsers, namely, Google Chrome and Opera. The energy consumed by the browser for loading the subjects is measured using the BatteryStats plugin of Android Runner. Two performance metrics are collected via the PerfumeJS plugin of Android Runner, namely, (i) First Contentful Paint (FCP) and (ii) Full Page Load Time (FPLT). The main results of the study are: (i) Ads significantly impact the energy consumption of mobile Web apps for both browsers, with a large effect size (see Fig. 7). (ii) Analytics have a significant impact on the energy consumption of Chrome, with a medium effect size (see Fig. 7), but not on Opera. (iii) Both ads and analytics do not significantly impact the FCP metric on both browsers. (iv) Both ads and analytics significantly impact the FPLT metric on both browsers, but with a small effect size. This study provides evidence that both ads and analytics do have a significant impact on the energy consumption and performance of mobile Web apps.

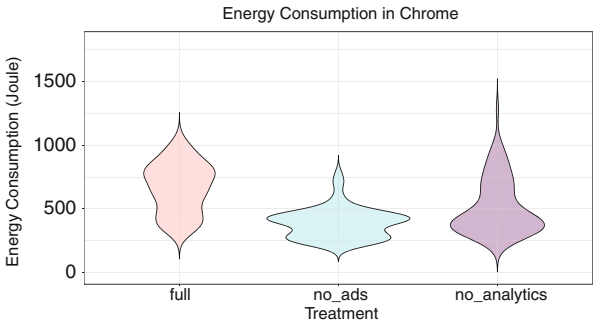


Fig. 7 Energy consumption of Web apps (full, without ads, and without analytics) (From [41])

Web developers are advised to limit both ads and analytics in their mobile Web apps to reduce their energy consumption and improve their loading time.

6.2 *Scientific Studies on Mobile Apps*

6.2.1 Comparison of the Energy Consumption and Performance of Native Android Apps and Their Web Counterparts [24]

This study compares the quality of the native version of a mobile application against its Web counterpart. This work has been (remotely) presented at MOBILESoft 2023 by a student, thus giving relevance to the commitment of the students and the quality of both the course and the lab. This study observes the problem mainly from a user perspective. If correctly informed, users may choose to use the native or the Web version of an application to save battery life and for better performance. Our daily lives have become more and more dependent on mobile applications. Music-streaming platforms, digital newspapers, and social networks are just a few examples of widely used mobile applications. On-demand content is usually provided through two versions of a mobile application: a native and a Web version. Native applications are designed for a specific system and have direct access to the hardware resources of the system. For example, native applications can be specifically designed for Android and fully support the GPS, camera, and accelerometer of the device. The Web version of the application, on the other hand, is portable and implemented using Web languages, e.g., HTML, CSS, and JavaScript. A set of ten well-known mobile applications is sampled from different categories, such as social networks and news. Each application is stressed with a typical usage scenario for that application. For example, the scenario executed to profile the news application of ESPN involves a user opening an article, scrolling down, and checking the next one. Both versions of the application are tested using default settings. A scenario is executed 25 times for each application, profiling the performance and the energy consumed per run. Figure 8 depicts the results of this study. The top row shows that native apps tend to be more energy-efficient than their Web counterparts. The most energy greedy applications are social media and e-commerce with Twitch as the most energy-consuming application. The Web version resulted in using considerably more CPU and memory, as shown by middle and bottom row of Fig. 8. Instead, there is no significant evidence that native applications have improved network traffic and frame time.

6.2.2 Runtime Efficiency of Android Apps Migrated to Kotlin [42]

This study is the result of a master's thesis carried out using the Green Lab tools, and its main characteristics is its *mixed-method* research design involving a GitHub

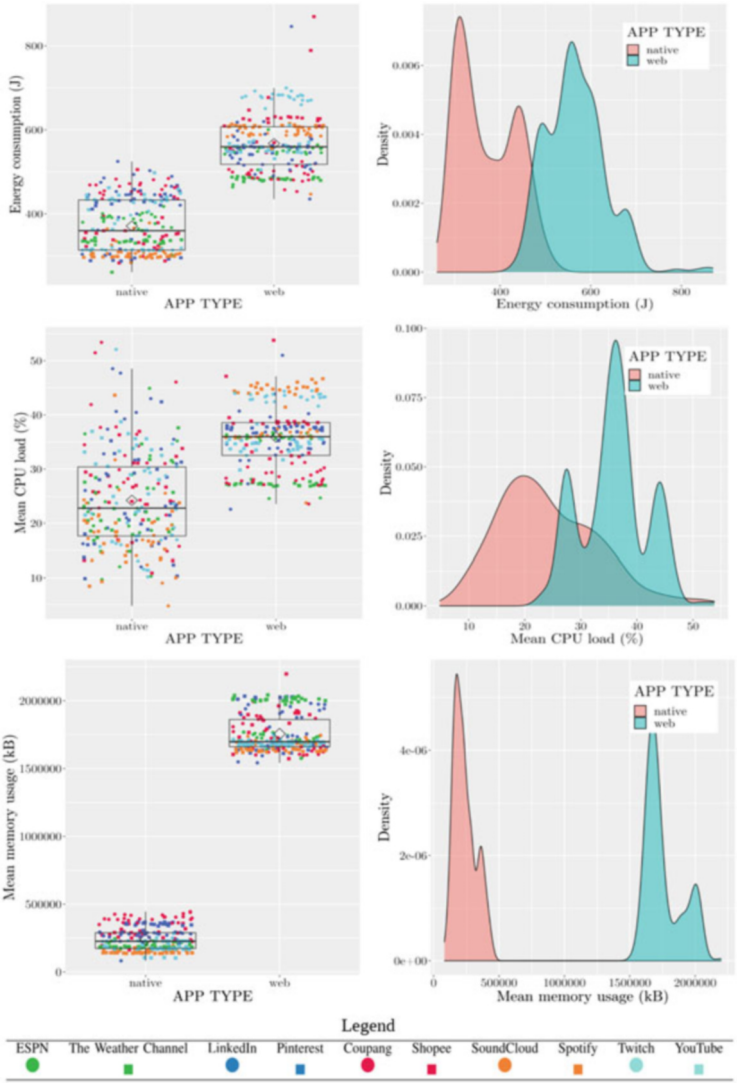


Fig. 8 Energy consumption (top row), CPU load (middle row), and memory usage (bottom row) for ten mobile applications (Adapted from [24])

mining phase and a measurement-based experiment. The study was awarded with the Distinguished Paper Award at IEEE SCAM 2021 (co-located with ICSME).

This study targets Android apps developed using Kotlin. Specifically, we aimed at empirically assessing the impact of the *migration* from Java to Kotlin on the runtime efficiency of Android apps. To achieve this goal, the student mined 7,972 GitHub repositories containing Android apps and identified 451 apps containing

Kotlin code. By applying a cross-language clone detection technique [11], they detected 62 commits that represent a full migration to Kotlin while keeping the app functionally equivalent. This mining phase confirmed that most open-source Android apps either fully migrated to Kotlin (>90% Kotlin code) or contained low portions of Kotlin code (<10%). After the mining phase, a sample of ten apps that fully migrated to Kotlin have been selected, and then the measurement-based experiment has been executed to compare their Java and Kotlin versions. The study considered six runtime efficiency metrics: CPU usage, memory usage, number of calls to the garbage collector, frame times, app size, and energy consumption. The experiment has been orchestrated via Android Runner and targeted two smartphones. By referring to Fig. 9, the results of the experiment highlight that migrating to Kotlin has a statistically significant impact on CPU usage, memory usage, and render duration of frames (albeit with a negligible effect size), whereas it does not impact significantly the number of calls to the garbage collector, the number of delayed frames, app size, and energy consumption. This study provides evidence that *developers can migrate their Android apps to Kotlin and expect comparable efficiency at runtime*.

6.2.3 Evolution of Kotlin Apps in Terms of Energy Consumption [2]

This a follow-up of the study described in Sect. 6.2.2. In this case, instead of looking at apps migrated to Kotlin, a team of Green Lab students performed a longitudinal analysis of the energy consumption of three selected Kotlin apps. The goal of this study is to empirically assess how the energy consumption of Kotlin applications evolves over time. Students first rigorously selected three open-source Kotlin applications via well-defined inclusion and exclusion criteria and a quality assessment procedure. Then, they cloned the GitHub repository of each app and selected a total of 57 combined releases of those apps. The energy consumption of each of those 57 releases is measured using Android Runner, and the collected measures are statistically analyzed in order to identify energy spikes and drops (see Fig. 10 for an example). Finally, source code changes occurring in "energy-relevant" releases are manually analyzed to identify possible causes for the observed energy fluctuations. The study confirms that the energy consumption of Kotlin apps generally follows a growing trend along releases, and in all apps, significant energy spikes and drops have been detected. Among the major factors that impacted those fluctuations, we noted that those co-occurred with OS upgrades, the release of new app features, the usage of poorly chosen design patterns and libraries, the injection of UI-related issues, and unstable app versions.

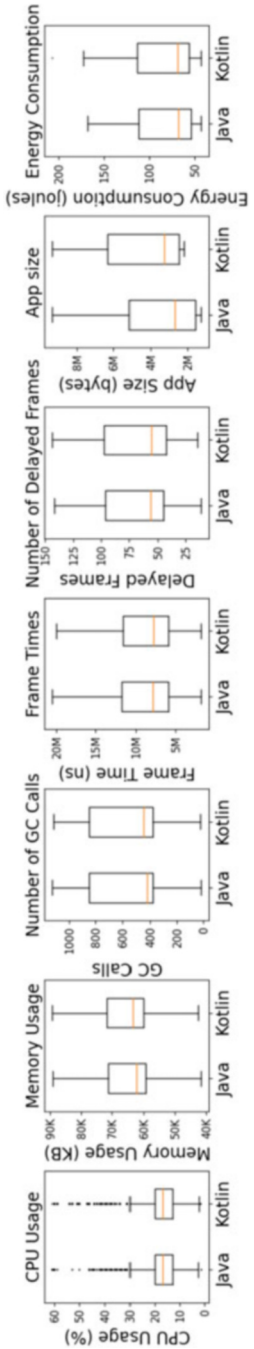


Fig. 9 Runtime efficiency of Java and Kotlin apps on Nexus 5 (Adapted from [42])

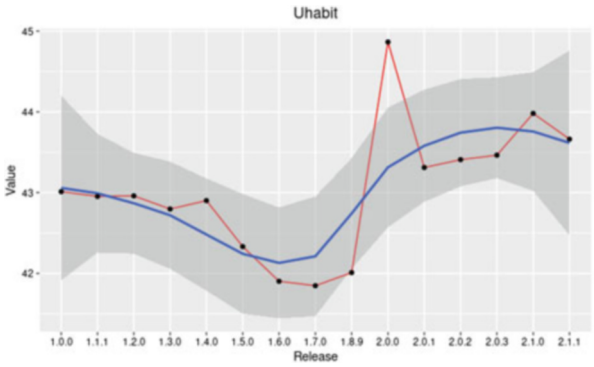


Fig. 10 Energy consumed by the uhabits app across its releases in J (red line = raw energy consumption; blue line = LOESS smoothed consumption; gray area = confidence interval 0.95) (From [2])

6.3 Scientific Studies in Other Domains

6.3.1 On the Energy Consumption and Performance of WebAssembly Binaries Across Programming Languages and Runtimes in IoT [65]

This work represents the first project regarding IoT in the Green Lab and extends the work presented at EASE 2022 [59]. This study analyzes the potential of WebAssembly (WASM) [54] in another application domain, i.e., IoT. One of the students, who also authored the paper, presented the work at EASE 2023 in Oulu, giving the right coronation of effort, as well as recognition and visibility to the students involved in the project and the Green Lab.

WASM allows software written using non-Web programming languages, such as C++ and Rust, to be executed inside a Web browser. Modern browsers can load WASM modules through JavaScript and execute them through a separate browser sandboxed environment. WASM showed potential for increased security, portability, and close-to-native performance. For this reason, WASM represents a good solution for fast deployable and updatable software for IoT devices. IoT involves running software on a plethora of resource-constrained devices. Devices of an IoT system can vary from a few to thousands and need to be regularly updated to add new features and fix bugs. The adoption of WASM on IoT devices is achieved by using runtime environments. A runtime environment works as an interface between a WASM-executable and the underlying system and provides basic system-related services, such as I/O operations.

In this study, we investigated the impact of different programming languages and runtime environments on the performance and energy consumption of WASM binaries in the context of IoT systems. We selected four different programming languages, namely, Rust, Go, JavaScript, and C, and two different WASM runtime environments, i.e., Wasmer and Wasmtime. The programming languages were evaluated using three software applications taken from the Computer Language Benchmark Game (CLBG). We considered the implementation of each software in each programming language. Each implementation was executed using Wasmer and

Wasmtime on a Raspberry Pi 3 Model B. The execution of the experiment resulted in 24 trials (4 programming languages × 3 benchmarks × 2 runtime environments), each one executed 10 times.

Rust is the most energy- and time-efficient language, while C is the most memory-efficient (see Fig. 11). However, JavaScript resulted to be the most resource-demanding language. From the analysis of runtime environments, Wasmer is found to be more efficient than Wasmtime in terms of performance and power consumption. We study also the interaction between the runtime environment and programming language that showed energy and time efficiency for all programming languages running using Wasmer. For memory usage, instead, the executions through Wasmer resulted worse than those on Wasmtime.

6.3.2 Computation Offloading for Ground Robotic Systems Communicating over Wi-Fi: An Empirical Exploration on Performance and Energy Trade-Offs [18]

This study shows how we exploited the Green Lab to perform experiments in the area of robotics software. We set up a test environment for a TurtleBot3 Burger that moves autonomously in space. This environment is used to evaluate the impact of different offloading strategies on energy consumption and the performance of the robot by diversifying the tasks performed by the robot.

Robotics system is known to be resource-constrained, which means resource utilization greatly impacts its operational time. For example, battery-powered robots need to optimize their energy consumption to ensure that they have enough power to accomplish their mission. To overcome hardware limitations, computation-intensive tasks can be offloaded to the Cloud. However, the integration of offloading mechanics within robotics software increases network utilization, which may reduce the quality of service and raise the energy consumption of the network.

This study empirically assesses the impact of offloading strategies on the performance and energy consumption of autonomous driving robots. The assessment involved two experiments. In the first experiment, our goal was to characterize offloading strategies and their impact on energy consumption and performance of the robot, while in the second, we analyzed the impact of the offloaded task parameter values on the same metrics. The robot subject to the experiment is a TurtleBot3 Burger [3], which performs three tasks: mapping (SLAM), navigation, and object recognition. These tasks are implemented using ROS. During the mapping task, the robot builds a map of the (unknown) environment in which it operates and localizes itself in the map. The navigation task, on the other hand, enables the autonomous driving of the robot to a goal location on a map, while object recognition consists of the identification of objects on the map by exploiting robot cameras. The tasks have different parameters, such as image resolution and frame rate for object recognition and the map for the navigation task that can be known or unknown. We observed the robot operating using different values of these parameters, referred in the paper as a configuration. Task parameter configurations

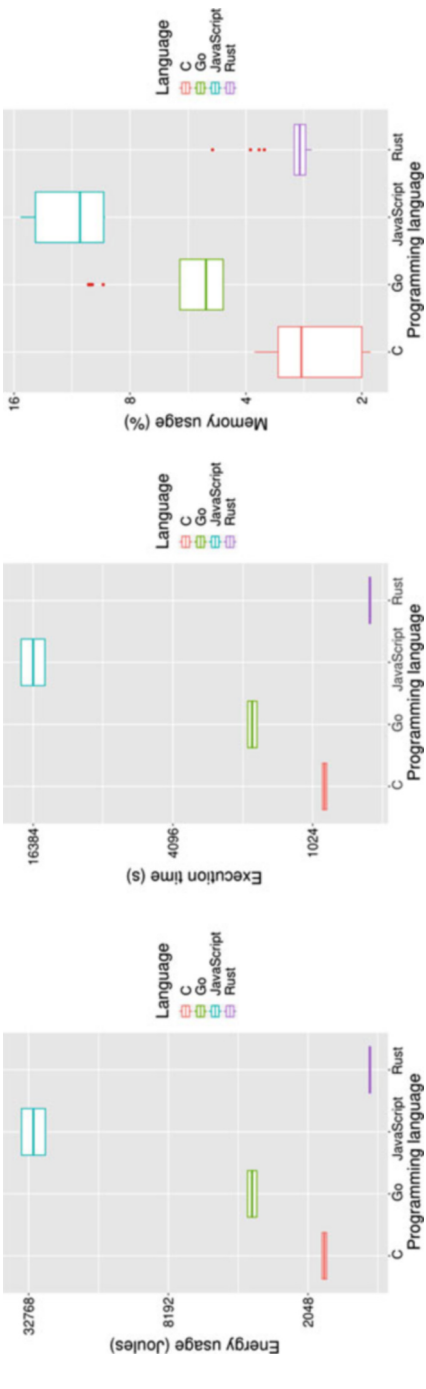


Fig. 11 Energy consumption and performance of WebAssembly binaries across programming languages (Adapted from [65])

Table 3 Task parameter configurations (Adapted from [18])

Factor	Parameter	Config. 1	Config. 2	Config. 3
Object recognition	Image resolution	640×480 px	640×480 px	1280×960px
	Image frame rate	20 fps	20 fps	60 fps
SLAM	Number of particles	–	5	30
	Temporal updates	–	Off	On
Navigation	Velocity samples	10×20	10×20	20×40
	Simulation time	1.5 s	1.5 s	3 s
	Map	Know	Unknown	Unknown

considered in this study are described in Table 3. The configurations have different magnitude, for example, Conf. 3 considers the highest value for each parameter. The TurtleBot3 is observed operating in a test environment. For each task, we performed a mission with and without offloading the selected task to a remote PC. In the first experiment, we launched the robot with Conf. 1 and Conf. 2, which differ in the parameter on map knowledge, while in the second, we executed the robot with Conf. 3. Both experiments involved nearly 100 trials with a total experimentation time of around 23 hours.

The results show that offloading reduces processing time for computationally intensive tasks, i.e., object recognition. There were no significant differences in performance on other tasks. The experiment highlights increasing latency due to higher network utilization. The optimization observed for processing time reflects energy consumption, which is reduced when offloading compute-intensive tasks. The analysis of task parameters shows their configuration has a critical impact on performance and energy consumption. Thus, tuning the parameters of ROS tasks is critical to achieving energy and performance efficiency in robotics systems.

6.3.3 Mining the ROS Ecosystem for Green Architectural Tactics in Robotics and an Empirical Evaluation [34]

In this study, we exploited the Green Lab to validate the results obtained after obtaining four architectural tactics for energy-efficient robotics software from the literature. The Green Lab, in this case, proved effective in transferring and validating knowledge about software engineering practices to real-world contexts.

Software is becoming central in robotics systems. For example, NASA uses more control software on its new exploration missions than it has on all of its previous missions combined. However, at present, robotic software is still poorly engineered, despite its increasing complexity and size. In addition, robotics software design focuses only on performance and functional aspects, leaving important issues such as energy consumption out of the design process. The integration of energy consumption analysis at design time would bring enormous benefits to the quality

of the final system. Battery-powered robots could adopt strategies at runtime to consume less energy and thus increase operational time.

This study identifies and empirically evaluates architectural tactics for robotics systems. The architectural tactics are identified by mining ROS-related data from open-source software repositories and Stack Overflow (SO). The collected dataset includes discussions on SO, GitHub pull requests, commit messages, and source code. After collecting the dataset, it is processed using a set of keywords related to energy concerns (e.g., green, battery, power). Resulting data are manually validated by researchers, which remove false positives and keep data concerning software architecture. The last filter applied to the collected data is a thematic analysis, in which the researcher manually grouped energy-related software architecture design decisions, e.g., usage of low power mode, and threshold-based mechanisms. This process resulted in four architectural tactics for energy-efficient robotics software:

- *EE1—Limit Task:* robot tasks, such as video streaming or robot navigation, can be limited when their energy consumption reaches a critical level.
- *EE2—Disable Hardware:* robot operation time can be extended by reducing unnecessary usage of hardware resources.
- *EE3—Energy-Aware Sampling:* sensors sampling frequency impacts the energy consumption of the robot. The sampling frequency of the sensors can be adjusted based on the energy level of the robot.
- *EE4—On-Demand Components:* robots components should be started only when needed and shut down when they are not required. This tactic saves energy consumed by components when not in use.

The tactics are empirically validated on a TurtleBot3 Burger [3]. The study evaluates the energy consumed by robotics when applying the tactics across different robot movement strategies and environments. The evaluation of the tactics involves a baseline (B) in which no tactics are applied and a combined treatment that involves the application of all the tactics simultaneously (C). The environments considered have obstacles or not, i.e., empty or cluttered, while the movement strategies consist of no motion; random movement in the environment, i.e., autonomous strategy; and following a prefixed plan, namely, sweep.

The results show that the application of the tactics improves energy efficiency. Moreover, combining all tactics results in greater energy efficiency than any single strategy in isolation. As shown in Fig. 12, the energy consumed by the robot is highly influenced by the movements of the robot and the physical environment in which the robot operates. This study highlights a fundamental problem for both researchers and robotics software developers: energy consumption. The results of the mining phase confirm that energy is infrequently discussed by roboticists. Moreover, this work represents a starting point for researchers who want to define and refine further architectural tactics for robotics software, which are proven to be beneficial for robots' energy efficiency.

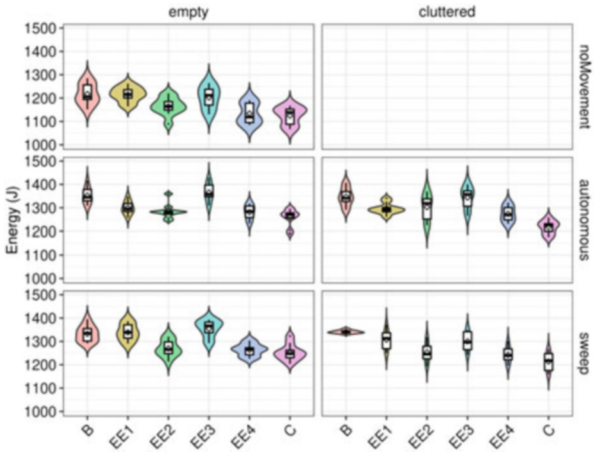


Fig. 12 Energy consumption across all movements strategies and environments (From [34])

7 Lessons Learned and Recommendations

The design of the Green Lab course described in Sect. 3 is the result of ten years of experience in the delivery of the course and can be reused as a basis for delivering similar courses in other universities. We reached this stage iteratively. Over the years, we piloted several alternative solutions and various educational components and refined the content of the course edition by edition. This resulted in an accumulation of a number of lessons learned that we would like to share with the community. We report such lessons learned and make recommendations for instructors in the remainder of this section.

7.1 General Recommendations

First of all, we learned that it is fundamental to **clarify expectations to students from day zero**. The strong research orientation of the course might not be common in some universities, and students might have a more didactic attitude concerning the course, instead of aiming to achieve scientific goals and solving open problems. In other words, students might be caught off guard when their project is evaluated with a critical eye from the instructors. As a solution, in the first lecture of the course, the instructors not only introduce the course (in terms of covered contents, learning objectives, assessment methods, and practical aspects) but also explicitly inform students that the teams whose project is well reported and scientifically valid will be invited to participate in a scientific publication based on their project. In addition, as homework for the first lecture, students are asked to watch a video where one of the instructors presents the results of an experiment emerging from the course (specifically, it is about the results of [36]).

Related to the previous point, **instructors should set high stakes for those teams that want to "chase the stars," but they shall still be open to teams that just want to pass the course**. As the instructors say in the first lecture, "this course is about opportunities"; however, it is important that students are not *required* to catch such opportunities. In each edition of the course, there have always been teams that just wanted to pass the course with a good grade and learn just enough about ESE methods, without spending extra effort for the scientific publication. We believe that this is fair to students and those students shall still be able to get the maximum grade for their project. At the end of the day, not all of our students are aiming to do a PhD in empirical software engineering, and as instructors, we must accept it.

Instructors should **keep the scientific publication not central to the project and value more students' learning than the scientific output**. This point is important to do not create too much pressure for students, and it should be considered as a good extra reward for them. Concretely, during the first lecture, students are asked to form their teams as soon as possible (recall that the deadline for the first assignment is generally at the end of the second week of the course), and each team is also asked to indicate if they are interested in contributing to a scientific publication based on their project (in the form, we also allow for a *"Maybe (if time allows)"* intermediate level). In any case, instructors' feedback and grading rubrics are exactly the same for all teams. Also, in previous editions of the course, it happened also that a team was not willing to pursue the scientific publication based on their project; in those (rare) cases, the instructors did not proceed with the submission.

In some past editions of the course, students raised concerns about the usefulness of the labs. We learned that in those cases, the labs were (i) focusing primarily on the basics of the tools and R and (ii) less interactive than in the other years (i.e., the teaching assistant was just presenting with slides and briefly carried out some practical tasks). The lesson learned here is that **labs shall be carefully planned in such a way that students can work on practical aspects related to their projects**. Higher lab interactivity can be achieved by (i) asking students to compute basic summary statistics on real data coming from previous projects, (ii) creating scripts about the contents of the labs beforehand and making sure that they contain tasks for students, and (iii) investing time in educating the teaching assistants about the technical aspects of the lab (more technically confident teaching assistants generally gave better labs).

Designing and running a course like the Green Lab takes a considerable amount of time. This investment must pay off for instructors. First of all, a course like the Green Lab is a **good opportunity for researchers to carry out research in the context of their education activities**; in a way, running the course is not a purely teaching activity, but it is a way for researchers to have education and research *intersect*, allowing instructors to continue doing their research activities in the weeks of the course (instead of putting them on hold). Moreover, by design, students who attend the Green Lab have the fundamentals of ESE methods in general (independently of the energy aspects). This means that those students are already able to design and conduct an empirical study in the context of, e.g., their final

project/thesis. Another strategy for capitalizing on this investment is to advertise the availability of further projects and theses during the course, allowing the instructors to have a **continuous research-oriented relationship with students even after the course**. For example, at VU Amsterdam, there are two additional research-oriented courses that students can put in their study plan (for a total of 18 credits) and a 30-credit final master's project; if a researcher and a student agree to have the Green Lab, those two courses, and the final project on the same research topic, they will work together for a total of 54 credits, which is the equivalent of more than 8 months' full time. As such, this creates an opportunity for both the student and the researcher to design a more meaningful and well-structured research project than those carried out in more compact time frames.

Over the years, we also noted that courses like the Green Lab can be used as a good training environment for discussing **ethical aspects** related to empirical software engineering and IT in general. First of all, by directly targeting the energy efficiency of software, issues related to the carbon footprint of IT are inherently central in all lectures of the course; this is also by design since the course is given as a module of the Software Engineering and Green IT master's program, and students enrolled in such program (i) expect this kind of discussions and (ii) have already a certain sensibility toward themes related to green software and sustainability. Moreover, by design, the course touches upon aspects related to *scientific integrity* and allows students to get a hands-on experience about the importance of (i) the replicability/reproducibility of scientific experiments (e.g., some projects are about replicating an already-published study starting from its replication package, and all student teams need to produce a complete replication package for their own projects) and (ii) properly reporting scientific studies (e.g., all teams need to honestly describe the threats to the validity of their experiment and to follow empirical best practice when writing the report of their project). During the three project assignments, the teaching team checks the ethical aspects of the experiments to ensure the reliability of the results and the claims derived from them. As the students evaluate the energy consumption of real software, their experiment may involve subjects and produce results that are sensitive to the stakeholders of the research, such as IT software developers, users, and IT companies.

As reported in Sect. 2, the duration of the Green Lab course is 2 months. We are aware that courses in other universities can have a longer duration. Below we indicate **how a course like the Green Lab might be expanded to longer courses**:

- In-depth coverage of other ESE research methods (e.g., repository mining, qualitative survey, systematic literature review) in addition to controlled experiments and integration of such research methods into students' projects.
- Expanding the (currently single) lecture on techniques for developing energy-efficient software, possibly integrating it with intermediate experiments providing evidence about the discussed techniques.

- Simulate a scientific conference by (i) setting up a reviewing system where students peer-review and discuss other teams' projects and (ii) teams present their results at the end of the course.
- Organize "replication sessions" where teams carry out an additional experiment focusing on replicating (and learning from) an experiment carried out by other research groups in the world (this avoids the bubble effect where students are primarily exposed to experiments of the S2 group at VU Amsterdam).
- Organize an intermediate hackathon on a selected common case (this might give a bonus to the winners of the hackathon).
- Carry out more hands-on labs and in-depth labs on various measurement tools (and different domains—IoT, mobile apps, servers, etc.).

7.2 Recommendations About Students' Projects

On top of experience and knowledge about ESE research methods, **the teaching team must have at least a basic knowledge of the technical/technological aspects of students' projects**. Given the relatively tight schedule of the course, the teaching team must be able to provide continuous (and timely) support to teams having technical issues during the course. Possible strategies to achieve this include (i) involving scientific assistants mentioned in Sect. 5 as teaching assistants within the course (in this way, it is the same people maintaining the tools supporting students in their usage); (ii) train teaching assistants on the technical aspects of the projects before the beginning of the course, e.g., by executing small-scale replications of previous experiments; or (iii) enrolling teaching assistants based on their technical skills and assigning them only projects on technologies related to such skills.

It is suggested to **create multiple project tracks for each edition of the course**, roughly about double the number of student teams. In the first edition of the Green Lab, we had only two teams of students, so it was natural to have only two project tracks for the whole course. Differently, in the last edition of the course, we had 19 teams of students. Having only two project tracks could have raised the likeliness of fraud among students, and it could have made the assessment process extremely boring and repetitive for the teaching team. Thus, we opted to provide a large number of project tracks to students (i.e., 30 individual tracks). Those project tracks touched upon different technologies (e.g., Android APIs, data compression algorithms, large language models), platforms (e.g., Web apps, VR apps, IoT, HPC), and types of research methods (e.g., longitudinal studies, correlation studies, observational studies). When students register their team, they also provide their level of preference for each proposed project track, and then it is the teaching team that assigns the track to each team accordingly. This proved successful since students could choose for their project those tracks that best fit their skills and wishes. It is important to note that, in order to avoid assigning low-voted tracks, *it is possible to assign the same project track to different teams*. If on one side this is a risk for the publication potential for the track (i.e., only one team in principle

will be able to publish its project), on the other side, it creates safe competition among teams; also, it is up to the teaching team to propose slight variations to the redundant tracks in order to make them complementary (and thus equally valuable). As an example, our study on WebAssembly binaries across programming languages and runtimes [59] has been performed also on Android.

Creating multiple project tracks every year is a time-consuming and difficult endeavor. So, it is suggested to potential instructors to **keep a log of potentially valid projects all year long**. Based on our experience, having a textual file with a 2-liner for each potential project track and some support links is more than enough. Then, at the beginning of the course, the teaching team meets, distributes the tracks among each other, and expands each 2-liner into one or two slides for each project track; those slides are then used in the first lecture to pitch project tracks to students. The list of potentially valid projects proved also to be a useful overview of interesting research directions in the area of energy-efficient software, independently of the Green Lab course. Examples of sources we recurrently use for potentially valid projects include (i) social media accounts of key tech engineers (e.g., Addy Osmani for Web performance), (ii) engineering blogs of relevant tech companies (e.g., Dropbox Tech Blog[20]), (iii) Web sites focusing on software development and technological advances like InfoQ[21] and Ars Technica,[22] (iv) scientific literature in software engineering conferences, (v) developers' discussion forums like Stack Overflow[23] in general or ROS Discourse[24] for robotics, and of course (v) daily discussions with peers and colleagues.

7.3 Recommendations on Tools and Equipment

Before delving into the details about tools and equipment, we reiterate that the Green Lab course is designed as a **technology-independent course**. In other words, the learning objectives of the course, its structure, assessment procedure, and the majority of lecture contents are exclusively about energy efficiency and ESE research methods, and not bound to any specific technology (e.g., Android, microservices, IoT) or application domain (e.g., finance, social media). The only technology-specific components of the course are the following: (i) the topics of students' projects; (ii) the first practical session on the lab environment, tools, and devices; and (iii) the examples the instructor uses during the lectures. Being technology-independent is important for us (and future instructors willing to reproduce the course) since in this way, the course does not need to be fully redesigned over the

[20] https://dropbox.tech
[21] https://www.infoq.com
[22] https://arstechnica.com
[23] https://stackoverflow.com
[24] https://discourse.ros.org/

years. Being technology independent, the course also allows students to acquire and apply ESE principles in other contexts, such as their final project (which might not even be about energy efficiency) or in their professional careers. Instructors willing to replicate the course in their own departments can reuse the course material as it is, with only minor adaptations, depending on available equipment (which might even be students' laptops). In any case, **we suggest instructors to build an inventory of the hardware (and software) equipment available in their department before running an edition of the Green Lab course**. In this way, students can still measure energy consumption in a correct and reliable manner, without the need to buy expensive hardware. This is also a good opportunity for collaborating with other research groups within a department/university, for example, by carrying out controlled experiments on the energy efficiency of software in the context of, e.g., robotics, high-performance computing infrastructures, and bioinformatics.

In case of a large number of students (e.g., above 50), **a good strategy to scale is to fix the technologies and used tools considered in the course**. For example, Android Runner was used by all student groups for their Green Lab projects starting from 2017/2018 until 2022/2023. This allows (i) teaching assistants to save considerable effort in preparing for labs and supporting students in case of technical issues and (ii) the teaching team to assess students' projects more efficiently. However, it is important to note that this also comes at the cost of flexibility when defining students' projects, thus potentially impacting the research output of the course. For example, in the years when we were using exclusively Android Runner, all students' projects were about mobile (Web) apps on Android. Moreover, we often reuse technologies that are proven to be effective for specific domains, saving time and resources when preparing experiments. In the last edition of the course, we decided to go beyond Android and cover multiple technologies and platforms again; this was possible thanks to the possibility of financially supporting three teaching assistants dedicated to the course. Besides using our open-source tools, such as Experiment Runner and Android Runner, we usually integrate, for example, Prometheus [58] and Grafana [9] within experiments involving microservice-based applications. Prometheus and Grafana are open-source tools enabling monitoring and visualization of a set of metrics, such as CPU utilization and energy consumption.

Tools pay off, but they require continuous maintenance; otherwise, they fire back. Over the years, we learned that just having a tool like Android Runner is not enough to ensure a successful execution of experiments. For example, in previous years, the Android Runner tool was tested on a relatively low number of smartphones, and students using it on their own devices were having several compatibility problems; in those years, we received some negative feedback from students about the fact that they were spending a lot of time in fixing issues with the tool, rather than learning how to conduct experiments on energy-efficient software. As a solution, we decided to allocate one teaching assistant for one day a week for the whole academic year whose main responsibilities are to (i) fix bugs in the tools used in the Green Lab, (ii) support students having issues with those tools, and (iii) implement new features in the tools as new needs arise. As a result, after

the injection of this new resource, we did not receive any negative feedback about tools in course evaluations, and students were able to focus on the main learning objectives of the course.

Green Lab equipment is shared between several users, mostly students and researchers. The equipment is continuously maintained to avoid being accidentally damaged after usage. **Students shall be carefully instructed about equipment usage, both hardware and software**. For example, we usually use a Monsoon power meter, described in Sect. 4, for experiments requiring high accuracy and that are executed on Raspberry Pis. The main channel of the Monsoon power meter needs to be connected to the GPIO pins of the Raspberry Pi, namely, a 5V source pin and a pin for the ground (GND). Users should be careful in supplying the correct voltage and avoid current spikes that can damage the Raspberry Pi. A protocol providing step-by-step instructions for using the equipment is crucial to avoid misuse. During our course, it also happened that students accidentally left software running on cluster nodes or used nodes concurrently. We tell students to stop all software that is launched after finishing their session and to check running processes before carrying out their experiment. Concurrent access is mitigated by using a shared calendar in which users book specific time slots on cluster nodes. Software installed on cluster nodes, smartphones, and development boards can also influence experiment execution, if not maintained. Different versions of tools and libraries can cause incompatibilities. Legacy versions of a tool or operating system can be less efficient and contain security flaws and, thus, influence experiment outcomes. Older versions of operating systems may not support newer versions of the software to be tested by users.

8 Conclusions

The main contribution of this chapter is the description of the Green Lab course, i.e., a course in the area of empirical software engineering in the context of energy-efficient software. The design, organization, and syllabus of the course can be used as a basis for delivering a similar course in other universities and programs. The provided online material should facilitate a replication of the course with relatively low effort. Our recommendations and lessons learned reported in Sect. 7 are the results of 10 years of experience in giving the Green Lab course and should help our peers in delivering an even better course than ours; several of those lessons learned may also apply to courses outside the ESE research area.

As future improvements of the course, we are evaluating the possibility of (i) integrating the usage of the Empirical standard [47], both as educational material and as an assessment tool (both for instructors and peer assessment), (ii) anticipating the discussion of threats to validity earlier [61] within the timeline of the course (before the deadline for Assignment 2), and (iii) complementing the Green Lab with a more general course about ESE research methods.

Finally, we invite the ESE community to adopt (or even improve) our open-source *tools* for supporting the execution of measurement-based experiments (see Sect. 5). Over the years, they allowed our research group and our students to save a *considerable* amount of time and to adhere by design to well-known empirical guidelines with very low effort; we are actively maintaining those tools, and we are open for feedback, comments, and possible collaborations about them.

Acknowledgments We would like to thank all students and teaching assistants of the Green Lab course (and its accompanying tools) over the years for having contributed to its development and various improvements. We would like to thank Giuseppe Procaccianti for having initiated this amazing and highly rewarding course and our industrial partners that donated technologies for the course initial setup. We would like to thank Massimiliano Di Penta, Marco Torchiano, and Giuseppe Procaccianti for having supported the course with original material (e.g., slides) we built upon when preparing some components of it. We also thank all guest speakers who contributed to the success of the course with their guest lectures. Finally, we thank the Computer Science Department of Vrije Universiteit Amsterdam for providing the organizational and financial support to run the course.

This project is partially supported by (i) the LETSGO Project, promoted by the Netherlands Enterprise Agency (Rijksdienst voor Ondernemend Nederland), and (ii) the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 871342 "uDEVOPS."

References

1. Adafruit: Adafruit ina219 current sensor breakout. https://learn.adafruit.com/adafruit-ina219-current-sensor-breakout/overview. Accessed 8 Dec 2023
2. Ahmed, H., Boshchenko, A., Khan, N., Knyajev, D., Garifollina, D., Scoccia, G., Martinez, M., Malavolta, I.: Evolution of Kotlin Apps in terms of energy consumption: an exploratory study. In: Proceedings of the International Conference on Information and Communications Technology for Sustainability (ICT4S), pp. 46–56. IEEE, Piscataway (2023)
3. Amsters, R., Slaets, P.: Turtlebot 3 as a robotics education platform. In: Robotics in Education: Current Research and Innovations 10, pp. 170–181. Springer, Berlin (2020)
4. Anderson, L.W., Krathwohl, D.R.: A taxonomy for learning, teaching, and assessing: a revision of Bloom's taxonomy of educational objectives: complete edition. Addison Wesley Longman (2001)
5. Ardito, L., Coppola, R., Morisio, M., Torchiano, M.: Methodological guidelines for measuring energy consumption of software applications. Sci. Program. **2019**, 1–16 (2019)
6. Basili, V.R., Caldiera, G., Rombach, H.D.: The goal question metric approach. In: Encyclopedia of Software Engineering. Wiley, London (1994)
7. Bloom, B.S., Krathwohl, D.R.: Taxonomy of Educational Objectives: The Classification of Educational Goals. Book 1, Cognitive Domain. Longman (2020)
8. Briand, L.C., Differding, C.M., Rombach, H.D.: Practical guidelines for measurement-based process improvement. Softw. Process Improve. Practice **2**(4), 253–280 (1996)
9. Chakraborty, M., Kundan, A.P.: Grafana. In: Monitoring Cloud-Native Applications: Lead Agile Operations Confidently Using Open Source Software, pp. 187–240. Springer, Berlin (2021)
10. Chan Jong Chu, K., Islam, T., Exposito, M., Sheombar, S., Valladares, C., Philippot, O., Grua, E., Malavolta, I.: Investigating the correlation between performance scores and energy consumption of mobile web apps. In: Proceedings of the International Conference on Evaluation and Assessment on Software Engineering (EASE), pp. 190–199. ACM, New York (2020)

11. Cheng, X., Peng, Z., Jiang, L., Zhong, H., Yu, H., Zhao, J.: Mining revision histories to detect cross-language clones without intermediates. In: Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, pp. 696–701 (2016)
12. Chowdhury, S., Di Nardo, S., Hindle, A., Jiang, Z.M.: An exploratory study on assessing the energy impact of logging on android applications. Empir. Softw. Eng. **23**, 1422–1456 (2018)
13. Colin King: Powerstat - a tool to measure power consumption. https://manpages.ubuntu.com/manpages/xenial/man8/powerstat.8.html. Accessed 12 Apr 2023
14. Cook, T.D., Campbell, D.T.: Quasi-experimentation: Design & Analysis Issues for Field Settings. Wadsworth Publishing Company (1979)
15. Cruz, L.: Tools to measure software energy consumption from your computer (2021). http://luiscruz.github.io/2021/07/20/measuring-energy.html. https://doi.org/10.6084/m9.figshare.19145549.v1. Blog post
16. de Oliveira Neto, F.G., Torkar, R., Feldt, R., Gren, L., Furia, C.A., Huang, Z.: Evolution of statistical analysis in empirical software engineering research: current state and steps forward. J. Syst. Softw. **156**, 246–267 (2019)
17. de Munk, O., Scoccia, G.L., Malavolta, I.: The state of the art in measurement-based experiments on the mobile web. Inform. Softw. Technol. **149**, 106944 (2022). https://doi.org/10.1016/j.infsof.2022.106944
18. Dordevic, M., Albonico, M., Lewis, G., Malavolta, I., Lago, P.: Computation offloading for ground robotic systems communicating over Wi-Fi—an empirical exploration on performance and energy trade-offs. Empir. Softw. Eng. **28**(140), 1573–7616 (2023). https://doi.org/10.1007/s10664-023-10351-6
19. Dybå, T., Kampenes, V.B., Sjøberg, D.I.: A systematic review of statistical power in software engineering experiments. Inform. Softw. Technol. **48**(8), 745–755 (2006)
20. Fudan Software Engineering Laboratory: Train Ticket Booking System. https://github.com/FudanSELab/train-ticket. Accessed 12 Apr 2023
21. Georgiou, S., Rizou, S., Spinellis, D.: Software development lifecycle for energy efficiency: techniques and tools. ACM Comput. Surv. **52**(4), 1–33 (2019)
22. Goldman, R.: Learning Proxmox VE. Packt Publishing (2016)
23. Gozalo-Brizuela, R., Garrido-Merchan, E.C.: ChatGPT is not all you need. A state of the art review of large generative ai models (2023). arXiv preprint arXiv:2301.04655
24. Horn, R., Lahnaoui, A., Reinoso, E., Peng, S., Isakov, V., Islam, T., Malavolta, I.: Native vs web apps: comparing the energy consumption and performance of android apps and their web counterparts. In: 10th IEEE/ACM International Conference on Mobile Software Engineering and Systems, pp. 44–54 (2023)
25. Intel: Rapl: Running average power limit energy reporting (2021). https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/advisory-guidance/running-average-power-limit-energy-reporting.html. Accessed 12 Aug 2023
26. Intel Corporation: Intel Power Gadget. https://www.intel.com/content/www/us/en/developer/articles/tool/power-gadget.html. Accessed 12 Apr 2023
27. Juristo, N., Moreno, A.M.: Basics of Software Engineering Experimentation. Springer, Berlin (2013)
28. Ko, A.J., LaToza, T.D., Burnett, M.M.: A practical guide to controlled experiments of software engineering tools with human participants. Empir. Softw. Eng. **20**, 110–141 (2015)
29. Lago, P.: A master program on engineering energy-aware software. In: 28th International Conference on Informatics for Environmental Protection (EnviroInfo), pp. 469–476. BIS Verlag (2014)
30. Le Pochat, V., Van Goethem, T., Tajalizadehkhoob, S., Korczyński, M., Joosen, W.: Tranco: A research-oriented top sites ranking hardened against manipulation. In: Proceedings of the 26th Annual Network and Distributed System Security Symposium, NDSS 2019 (2019). https://doi.org/10.14722/ndss.2019.23386
31. Linares-Vásquez, M., Bavota, G., Bernal-Cárdenas, C., Di Penta, M., Oliveto, R., Poshyvanyk, D.: Api change and fault proneness: a threat to the success of android apps. In: Proceedings of the 2013 9th Joint Meeting On Foundations of Software Engineering, pp. 477–487 (2013)

32. Linux Tools: perf - Linux profiling with performance counters. https://perf.wiki.kernel.org/index.php/Main_Page. Accessed 12 Apr 2023
33. Lutz, M.: Learning Python: Powerful Object-Oriented Programming. O'Reilly Media (2013)
34. Malavolta, I., Chinnappan, K., Swanborn, S., Lewis, G., Lago, P.: Mining the ROS ecosystem for green architectural tactics in robotics and an empirical evaluation. In: Proceedings of the 18th International Conference on Mining Software Repositories, MSR, pp. 300–311. ACM, New York (2021)
35. Malavolta, I., Grua, E.M., Lam, C.Y., de Vries, R., Tan, F., Zielinski, E., Peters, M., Kaandorp, L.: A framework for the automatic execution of measurement-based experiments on android devices. In: 35th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW '20), pp. 61–66. ACM, New York (2020)
36. Malavolta, I., Procaccianti, G., Noorland, P., Vukmirovic, P.: Assessing the impact of service workers on the energy efficiency of progressive web apps. In: Proceedings of the International Conference on Mobile Software Engineering and Systems, MOBILESoft '17, Buenos Aires, Argentina, May, 2017, pp. 35–45 (2017). https://doi.org/10.1109/MOBILESoft.2017.7
37. Mann, S., Muller, L., Davis, J., Roda, C., Young, A.: Computing and sustainability: evaluating resources for educators. SIGCSE Bull. **41**(4), 144–155 (2010). https://doi.org/10.1145/1709424.1709459
38. Manotas, I., Bird, C., Zhang, R., Shepherd, D., Jaspan, C., Sadowski, C., Pollock, L., Clause, J.: An empirical study of practitioners' perspectives on green software engineering. In: Proceedings of the 38th International Conference on Software Engineering, pp. 237–248 (2016)
39. Monsoon Solutions: Monsoon power monitor. https://www.msoon.com/. Accessed 8 Dec 2023
40. Murugesan: Harnessing green IT: principles and practices. IT Professional **10**(1), 24–33 (2008). https://doi.org/10.1109/MITP.2008.10
41. Petalotis, C., Krumpak, L., Floroiu, M., Ahmad, L., Athreya, S., Malavolta, I.: An empirical study on the performance and energy costs of ads and analytics in mobile web apps. Inform. Softw. Technol. **166**, 107370 (2024). https://doi.org/10.1016/j.infsof.2023.107370
42. Peters, M., Scoccia, G.L., Malavolta, I.: How does migrating to Kotlin impact the run-time efficiency of android apps? In: 21st IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM), pp. 36–46 (2021). https://doi.org/10.1109/SCAM52516.2021.00014
43. Pfleeger, S.L.: Experimental design and analysis in software engineering. Ann. Softw. Eng. **1**(1), 219–253 (1995)
44. Pinto, G., Castor, F.: Energy efficiency: a new concern for application software developers. Commun. ACM **60**(12), 68–75 (2017)
45. Pramanik, P.K.D., Sinhababu, N., Mukherjee, B., Padmanaban, S., Maity, A., Upadhyaya, B.K., Holm-Nielsen, J.B., Choudhury, P.: Power consumption analysis, measurement, management, and issues: a state-of-the-art review of smartphone battery and energy usage. IEEE Access **7**, 182113–182172 (2019)
46. Procaccianti, G., Lago, P., Vetro, A., Fernández, D.M., Wieringa, R.: The green lab: experimentation in software energy efficiency. In: Proceedings of the 37th International Conference on Software Engineering-Volume 2, pp. 941–942 (2015). https://doi.org/10.1109/ICSE.2015.297
47. Ralph, P., Ali, N.b., Baltes, S., Bianculli, D., Diaz, J., Dittrich, Y., Ernst, N., Felderer, M., Feldt, R., Filieri, A., et al.: Empirical standards for software engineering research (2020). arXiv preprint arXiv:2010.03525
48. Rittal: Rittal Power Distribution Unit. https://www.rittal.com/com-en/products/PG0800ITINFRA1/PG7274ITINFRA1/PGR11260ITINFRA1. Accessed 12 Apr 2023
49. Santos, E.A., McLean, C., Solinas, C., Hindle, A.: How does docker affect energy consumption? Evaluating workloads in and out of docker containers. J. Syst. Softw. **146**, 14–25 (2018)
50. Saraiva, J., Zong, Z., Pereira, R.: Bringing green software to computer science curriculum: perspectives from researchers and educators. In: Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education, vol. 1, pp. 498–504 (2021)

51. Seltman, H.J.: Experimental Design and Analysis. Book Is on the World Wide Web (2018)
52. Sheskin, D.J.: Handbook of Parametric and Nonparametric Statistical Procedures. Chapman and Hall/CRC (2003)
53. Shull, F., Singer, J., Sjøberg, D.I.: Guide to Advanced Empirical Software Engineering. Springer, Berlin (2007)
54. Sletten, B.: WebAssembly: The Definitive Guide. O'Reilly Media (2021)
55. Supplementary material for the Green Lab course. https://zenodo.org/doi/10.5281/zenodo.11544897
56. Swanborn, S., Malavolta, I.: Robot runner: a tool for automatically executing experiments on robotics software. In: Proceedings of the ACM/IEEE 43rd International Conference on Software Engineering, pp. 33–36. ACM, New York (2021)
57. Tracking Clean Energy Progress 2023—Analysis - IEA (2023). https://www.iea.org/reports/tracking-clean-energy-progress-2023. Accessed 5 Dec 2023
58. Turnbull, J.: Monitoring with Prometheus. Turnbull Press (2018)
59. van Hasselt, M., Huijzendveld, K., Noort, N., de Ruijter, S., Islam, T., Malavolta, I.: Comparing the energy efficiency of webassembly and JavaScript in web applications on android mobile devices. In: Proceedings of the International Conference on Evaluation and Assessment on Software Engineering (EASE), pp. 140–149. ACM, New York (2022)
60. Vegas, S.: Analyzing software engineering experiments: everything you always wanted to know but were afraid to ask. In: Proceedings of the 39th International Conference on Software Engineering Companion, pp. 513–514. IEEE Press (2017)
61. Verdecchia, R., Engström, E., Lago, P., Runeson, P., Song, Q.: Threats to validity in software engineering research: a critical reflection. Inform. Softw. Technol. **164**, 107329 (2023)
62. Verdecchia, R., Lago, P., Ebert, C., De Vries, C.: Green IT and green software. IEEE Softw. **38**(6), 7–15 (2021)
63. Verdecchia, R., Sallou, J., Cruz, L.: A Systematic Review of Green AI. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, p. e1507 (2023)
64. de Vries, A.: The growing energy footprint of artificial intelligence. Joule **7**(10), 2191–2194 (2023)
65. Wagner, L., Mayer, M., Marino, A., Nezhad, A., Zwaan, H., Malavolta, I.: On the energy consumption and performance of webassembly binaries across programming languages and runtimes in IoT. In: Proceedings of the 9th International Conference on Evaluation and Assessment on Software Engineering (EASE), pp. 72–82. IEEE, Piscataway (2023)
66. WattsUp: Watts up? pro power monitor. https://github.com/isaaclino/wattsup. Accessed 5 Apr 2023
67. Weaveworks, C.: Sock shop–a microservices demo application. Retrieved September **5**, 2019 (2018)
68. Wickham, H., Çetinkaya-Rundel, M., Grolemund, G.: R for Data Science. O'Reilly Media (2023)
69. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering. Springer, Berlin (2012)
70. Yoo, A.B., Jette, M.A., Grondona, M.: Slurm: Simple Linux utility for resource management. In: Feitelson, D., Rudolph, L., Schwiegelshohn, U. (eds.) Job Scheduling Strategies for Parallel Processing, pp. 44–60. Springer, Berlin, Heidelberg (2003)

Experiences in Using the V-Model as a Framework for Applied Doctoral Research

Rodrigo Falcão, Andreas Jedlitschka, Frank Elberzhager, and Dieter Rombach

Abstract The pervasive role played by software in virtually all industries has fostered ever-increasing development of applied research in software engineering. In this chapter, we contribute our experience in using the V-Model as a framework for teaching how to conduct applied research in empirical software engineering. The foundational idea of using the V-Model is presented, and guidance for using it to frame the research is provided. Furthermore, we show how the framework has been instantiated throughout nearly two decades of PhD theses done at the University of Kaiserslautern (RPTU Kaiserslautern) in partnership with Fraunhofer IESE, including the most frequent usage patterns, how the different empirical methods fit into the framework, and the lessons we have learned from this experience.

1 Introduction

In recent decades, we have witnessed the rise and consolidation of empiricism as a key aspect of software engineering research. With its roots in early (and rare) initiatives between the mid-1960s and mid-1970s, empirical software engineering (ESE) has become an established and relevant field for researchers and practitioners. Over time, the teaching of ESE has found its way into universities, especially in graduate programs.

Education in empirical software engineering in graduate courses often focuses on the careful teaching of empirical strategies. For example, when the focus of a given course is on controlled experiments, its syllabus will comprise the introduction of basic concepts such as the meaning and purpose of experimental studies, variables,

R. Falcão (✉) · A. Jedlitschka · F. Elberzhager
Fraunhofer IESE, Kaiserslautern, Germany
e-mail: rodrigo.falcao@iese.fraunhofer.de; andreas.jedlitschka@iese.fraunhofer.de; frank.elberzhager@iese.fraunhofer.de

D. Rombach
RPTU Kaiserslautern, Kaiserslautern, Germany
e-mail: dieter.rombach@siak-kl.com

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2024

D. Mendez et al. (eds.), *Handbook on Teaching Empirical Software Engineering*,
https://doi.org/10.1007/978-3-031-71769-7_9

hypothesis testing, types of experimental design, the experimental procedure, and the application of statistical methods.

However, conducting graduate-level research requires students to have the ability to not only master empirical strategies. Empiricism is part of a broader endeavor, namely, research itself, which demands identification of an adequate problem, declaration of a research question, definition of general and specific goals, and so forth. Structuring software engineering research that uses the empirical approach and navigating through its phases are not trivial, especially in PhD-level research, where more rigor is required in all phases of research and a multi-method strategy is often needed. Furthermore, conducting and supporting the development of software engineering PhD theses in an applied research setting contribute another variable to the structuring of the research, for the research must address not only a scientifically important problem but also one with practical relevance.

In this chapter, we report on how the V-Model can guide teaching and performing applied research using the empirical approach. The V-Model is a conceptual model that provides a framework for a certain set of elements in a given context. These elements are organized in the shape of a "V," where the first element takes place at the top-left point of the shape. It is followed by other elements along the left side until the bottom point and then by the last elements along the right side of the shape. The raison d'être for the shape of the V-Model is that the elements featured on the right side map back to elements on the left side. In RPTU Kaiserslautern, the V-Model has been used as *a virtual model for products and for empiricism*. The virtual product model abstracts from what concrete processes have in common to fit any process; likewise, the virtual empirical model creates a frame that abstracts from concrete empirical strategies.

This chapter is the result of nearly 20 years of using the approach in the context of PhD theses done at RPTU Kaiserslautern in cooperation with Fraunhofer IESE.[1] Approximately half of the theses published between 2005 and 2023 used the V-Model explicitly (and an even higher number used it as a reference model, although it may not have been made explicit in some theses). To prepare this chapter, we reviewed these theses (23 in total), extracting usage patterns of the V-Model. Furthermore, we invited the authors of these theses to share the lessons they learned using the V-Model in their research. The feedback we received (from twelve authors) has been integrated into this text.

The remainder of this chapter is organized as follows: Sect. 2 describes the origins of the V-Model and where the ideas to adapt it to the classroom came from; Sect. 3 introduces the framework; Sect. 4 discusses the main variations in which the framework has been used; Sect. 5 reflects on the lessons we have learned; and Sect. 6 provides further reading recommendations.

[1] The list of books in the series "PhD Theses in Experimental Software Engineering," whose review is the main source for this chapter, can be found at https://s.fhg.de/phd-theses-in-ESE.

2 Background

The V-Model has its origins in the software engineering field but was neither initially intended to support applied research nor related to ESE. We have used the V-Model for this purpose as we found it an adequate vessel to bring our experience in applied research to the classroom.

2.1 The Origins of the V-Model

The introduction of the V-Model dates back to 1986 and was first featured in the Software Engineering Journal [24] in the context of the software testing process. It extends the waterfall process, relating requirements and design activities on the left side to validation activities on the right side. Figure 1 illustrates one of many variations of the original V-Model.

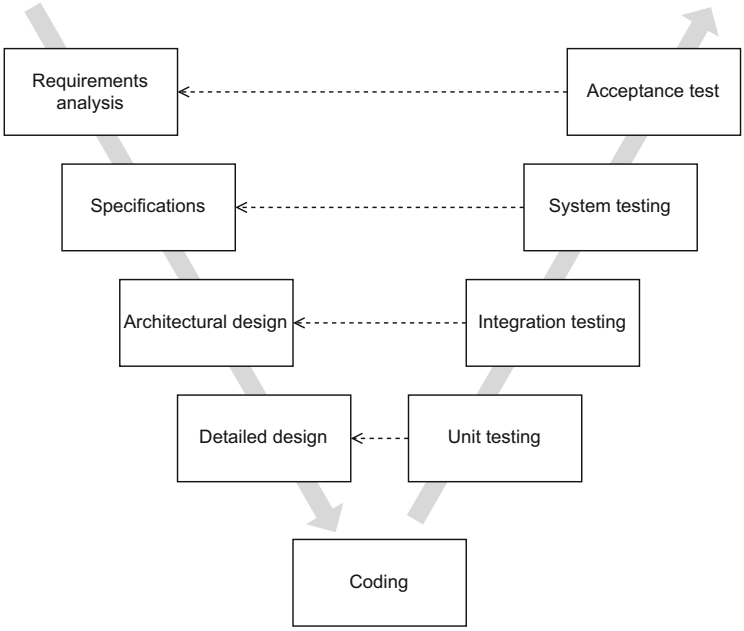


Fig. 1 A variant of the original V-Model being used to frame a traditional software development life cycle (Adapted from [18])

2.2 Bringing the Applied Research Approach to the Classroom

While the V-Model was originally introduced as a *process model*, this is not the only way it can be used. In the early 1990s, the fourth author introduced the V-Model as a *product model* and, shortly later, as an *empirical model* while designing a software engineering program focused on processes and empiricism at RPTU Kaiserslautern. When used as a product model, the V-Model is explicitly non-sequential. The idea was that any product being built through any process (be it sequential or iterative) should be consistent with the model. Therefore, the V-Model as a product model is a process-independent tool useful for building products.

When used as an empirical model (which has been the case in the PhD theses and, therefore, is on focus in this chapter), the V-Model is useful for process engineering, i.e., engineers benefit from its results. The principles behind the empirical V-Model have been used previously in applied research projects and are still being used now. At Fraunhofer IESE, the empirical research process is described as a cycle [11] with the following steps: *characterize the problem*, *set research goals*, *choose process for implementing the strategy*, *execute the research as planned*, *analyze the results*, and *package the results*. The principles underlying the process have been brought to the classroom with two adaptations. The first was that while it may be sufficient in certain industry projects to characterize the practical problem, in the academic environment, the scientific problem should always be evident. The second was that the process was designed as a cycle, implying a continuous improvement process, whereas in the educational context, a well-defined scoping strategy that explicitly points to an end yields greater benefits. For this purpose, the V-Model has been introduced as an instrument to bring the principles of the empirical approach to applied research to the lectures and dissertations in the graduate courses of RPTU Kaiserslautern.

Tip: Focusing on Principles

When using the V-Model as a product model, the focus is on providing students with stable principles. These principles should be valid, independent of the process to be used. Processes can change, and the way you build software varies, but in the end, there will be a clear problem, a set of requirements, a design, a solution, etc., and it does not matter whether these were generated sequentially or iteratively. This stability of principles ensures the reliability of the V-Model as a framework for teaching software engineering processes. The same is valid for using the V-Model as a framework for using the empirical approach in applied research in software engineering, independent of the empirical methods used.

3 The V-Model Framing Approach

To frame applied software engineering research with the support of empirical methods, we have, in most cases, used a five-stage V-Model. The starting point is the *practical problem*, followed by the *scientific problem*, the *solution*, the *internal validation*, and the *external validation*. The meta-model of the empirical V-Model is illustrated in Fig. 2.

Tip: Pre-requisites

While, in principle, there is no particular prerequisite to understanding the V-Model as a framework for ESE, it might make sense to introduce it in the context of an ESE course, where students should become familiar with the empirical approach in software engineering, because putting the V-Model framing approach in practice will require previous knowledge on, at least, the major empirical methods (case studies, surveys, and controlled experiments).

3.1 The Practical Problem

In an applied research setting, the research shall address real/practical problems, i.e., a problem that is known and experienced by practitioners. The practical problem—also referred to in the V-Model as "industry problem" or "state of the practice"—is usually illustrated through an example. In fact, from our experience, this is the typical starting point: describing one or more real scenarios where the problem is

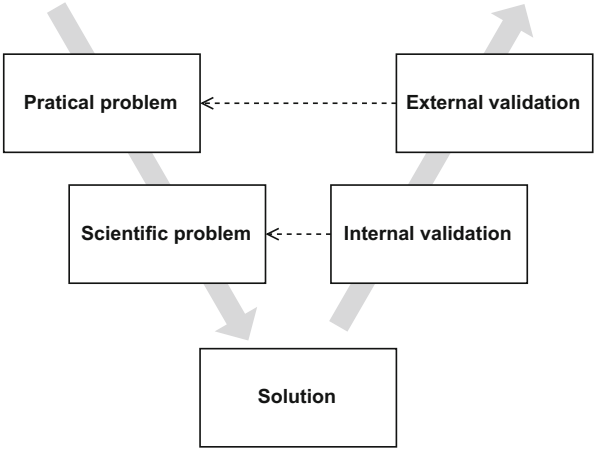


Fig. 2 The meta-model of the empirical V-Model

contextualized and explained. While we may start from a scenario description, at the end, we need one or more clear and concise *problem statements*. Therefore, the general question at this stage in the V-Model is:

? What Is the Practical Problem Faced by Practitioners?

At times, the practical problem may include one or more hypotheses about a potential improvement. For instance, in a given context, the problem may be the high costs of keeping software architecture documentation up to date. One may ask what could be done to decrease the maintenance effort, for example, by 20%.[2] At this point, an implicit hypothesis about a potential solution (which is still unknown and, therefore, to be devised) is that such a solution would achieve the improvement goal. At other times, there might be implicit hypotheses about the problem statement. Is the problem indeed a problem, or would it be someone's biased opinion? These are things to be investigated empirically.

Example (Elicitation of Context-Aware Functionalities[3])
Context-aware functionalities are functionalities that consider context to produce a certain system behavior, typically an adaptation or a recommendation. They are a desired feature in many software-based systems. However, they are hard to discover because they deal with implicit input. This is even more evident when trying to discover *unexpected* context-aware functionalities that aim to delight users by making the application exhibit what can be called "smart behaviors." However, the very fact that these functionalities are unexpected implies that they are not trivial to identify. In a survey with practitioners, researchers found that although some context modeling techniques exist, practitioners do not use them to support the elicitation of context-aware functionalities. Practitioners perceive context modeling as a complex activity and have therefore been overlooking it. It is particularly hard to figure out which contextual factors—individually or in combination with each other—are relevant for a given user task of interest. As a result, opportunities for describing and implementing context-aware functionalities are missed.

[2] Quantifiable improvement goals are usually preferable when it comes to hypotheses related to the practical problem, because companies want to know whether it is worthwhile to invest in a certain change.

[3] The running example used throughout this chapter refers to [7].

Empiricism for Investigating Practical Problems

The investigation of the practical problem may serve different purposes:

- **Providing evidence for the practical problem:** While the practical problem may sound reasonable when described in a certain scenario with a proper example, an empirical investigation can provide evidence about the existence and relevance of the problem. As elaborated by Jedlitschka and Pfahl [12] in their guidelines for reporting controlled experiments, it is important to declare what the problem is, where it occurs, who has observed it, and why it is important to be solved.
- **Exploring details of the problem:** Empirical studies can be used to test hypotheses about the problem and help pinpoint the root causes of the problem from a practical point of view. Furthermore, empirical investigation can help understand the size of the problem and measure its impact.
- **Explaining the causes of the problem:** When the researchers have hypotheses about the causes of the problem, these can be examined using empirical methods.

In our experience, one of the most frequent empirical methods employed to investigate the practical problem is **surveys with practitioners** (examples of such usage can be found in [1, 7, 25, 31]). Surveys can be implemented in different ways. For instance, when the aim is to investigate problems from a general population, surveys are frequently implemented using online questionnaires as the data collection strategy, as this can help reach more potential participants. On the other hand, when research is done in the context of a certain company, expert interviews are a common option (e.g., [19]). For a comprehensive introduction to surveys in software engineering, we refer to a series of articles by Kitchenham and Pfleeger [16].

Besides surveys, **case studies** are another typical empirical method used to investigate the practical problem (an example can be found in [17]). Depending on the research context, one or more case studies may be conducted to draw conclusions about the practical problem. Guidelines on case studies in software engineering are provided by Runeson et al. [28].

Even if the empirical investigation of the practical problem is done in the context of one single company, the risk of the investigated problem being only relevant in the context of the given company should not jeopardize the research, for the next step in the V-Model is to identify the scientific problem that underlies the practical problem.

3.2 The Scientific Problem

While the practical problem refers to the state of the practice, the scientific problem is concerned with the state of the art. The goal at this stage in the V-Model is to identify the scientific problem (also referred to as "research problem" or "research challenge") that underlies the practical problem. The general question between the identification of the practical problem and the identification of the scientific problem is:

? How Does the State of the Art Address the Problem?

If an adequate solution already exists in the state of the art, the immediate follow-up question to be answered is, *Why has the existing state-of-the-art solution not been used?* There may be a transfer issue. It may be the case, for example, that practitioners lack proper guidance, methods, or tools to benefit from state-of-the-art solutions. Conversely, if an adequate solution is not yet available, the gap in the state of the art becomes self-evident.

Based on the problem, one option is to explore the solution space, looking for existing alternatives with no assumptions about potential solution ideas. Existing solutions, when found, are accompanied by their limitations, which may provide hints for uncovering the scientific problem. Alternatively, it is possible to explore the solution space not only based on the problem but also with one implicit hypothesis (or maybe more) about a candidate solution: Why has *a certain approach* not been used to address the problem? *Has a certain approach been used?* The investigation may lead to finding a complete absence of evidence about the application of the approach or to identifying the limitations found by other researchers when trying to use the intended approach.

Example (Discovering State-of-the-Art Context Modeling Approaches) The researchers involved in the previous example had the idea of automating context modeling using a *data-driven approach*. So they reviewed the literature to search for existing solutions that followed the same idea. Although some approaches exist, they do not provide a comprehensive end-to-end solution to automate context modeling for eliciting context-aware functionalities. Among other things, researchers found that, in some cases, no process or conceptual model is provided. In other cases, no context model is explicitly generated; the solution is coupled to a particular data analysis algorithm or context model; or the ultimate goal of the approach is not the same. In other words, there is a lack of systematic end-to-end approaches for data-driven context modeling for the elicitation of context-aware functionalities.

Empiricism for Investigating Scientific Problems

The investigation of the scientific problem aims at:

- **Declaring the scientific problem:** The goal is to pinpoint the concrete, well-defined scientific problem that underlies the practical problem of interest.
- **Providing evidence for the relevance of the scientific problem:** The scientific problem must be worth investigating and must therefore be an open problem (i.e., not yet solved).
- **Explaining the root causes of the problem:** When the researchers have hypotheses about the root causes of the scientific problem, they can investigate them empirically.
- **Exploring potential solution strategies:** Previous work can provide directions about both promising and discouraging research directions in terms of solutions.

Systematic literature reviews (SLRs)—as proposed by Kitchenham and Charters [15]—have been used frequently as a way to gather evidence about scientific problems in software engineering. Being not a primary study type, systematic literature reviews can accumulate both positive and negative evidence (including, but not limited to, empirical evidence) about existing scientific problems, where positive evidence refers to the previous acknowledgment and documentation of these problems and negative evidence means the absence of reports in the literature about a particular or any general solution for the problems. Besides SLRs, mapping studies are also useful to review the state of the art. Guidelines for this strategy have been proposed by Petersen et al. [23].

Sometimes, the starting point may be a certain fact about a phenomenon observed in practice. For example, a certain software development process *X* may be found to be more efficient than an alternative software development process *Y* in Company A; however, in Company B, process *Y* is found to be more efficient than *X*. If the fact can be observed but the reason is unclear, explanatory research can be carried out to search for cause-effect relationships that might explain the root cause of the problem. In such cases, one or more **controlled experiments** can be used to support the investigation.

Once the scientific problem has been identified, the next step is to devise a solution for the problem.

3.3 The Solution

At the bottom of the V-Model, the design and implementation of a solution for the problem are a creative activity undertaken by the researchers. The guiding questions between the scientific problem and the solution are:

? What Is the Research Goal, and What Solution Strategy Can Be Used to Achieve It?

The characteristics of the solution will be specific to the problem under investigation. The solution may contribute to the foundations of the software engineering field to which it belongs (e.g., by providing and/or organizing concepts), to the methods used in the field (by describing processes or providing guidance), and/or to the technology used in the field (e.g., through the implementation of tools/automation). In any case, at this point, a solution is planned and implemented, be it a model, a method, a tool, etc. or a combination of them.

Example (Developing a Data-Driven Context Modeling Framework)
In the absence of a comprehensive approach, the data-driven context modeling idea was refined, and raising the level of abstraction, researchers decided to describe and implement a *framework*. The framework comprises a conceptual model explaining entities and relationships in the domain of data-driven context modeling and a process providing step-by-step guidance for what must be done. The framework itself was designed to be decoupled from any particular implementation of the contextual model or the data analysis algorithm. In order to apply the data-driven approach, the framework was instantiated through the development of a concrete context meta-model and a concrete data analysis algorithm.

Empiricism While Creating the Solution

In our experience, empirical methods are not typically *applied* at the solution stage (although there have been exceptions). However, *empirical work* does happen at the solution stage as it anticipates the planning of the next stages (internal and external validation). Empiricism at the solution stage means:

- **Defining hypotheses about the solution:** The solution is expected to improve in some way the as-is situation described through the practical and the scientific problems.

- **Outlining the evaluation:** The hypotheses must be verified using one or more empirical methods, which should be chosen in advance.
- **Performing pre-evaluation of the solution:** Before bringing the solution to the formal validation stages, preliminary validation studies can be performed with the support of empirical methods to provide early feedback and help improve the shape of the final solution.

In the ideal case, hypotheses about the solution should be specified right after the definition of the solution idea, for the implementation of the solution idea should be driven (and eventually checked) through the lens of the improvement goals implied (or explicitly mentioned) in the hypothesis.

Having defined the hypothesis and the solution strategy, the evaluation plan starts to take shape. Empirical methods for evaluating different aspects of the solution concepts in different contexts from different perspectives should be considered and chosen according to the needs and constraints of the research. From our experience, in most cases, the detailed plan for the internal and external validations can wait until the next stages (internal validation and external validation) to be formulated. In some cases, however, an initial evaluation is performed during the solution stage. It can take a variety of forms, ranging from a series of less formal feedback-gathering exercises such as an **early application with industry practitioners** (see [21] for an example) to **pilot tests** ("try runs"). In the latter case, a complete experimental procedure may be prepared in advance: through pilot tests, the procedure can be evaluated and adapted for the experiment to be carried out in the next stages. Furthermore, pilots help understand how the solution is perceived by others, even if it is still a partial solution.

Regardless of whether a preliminary evaluation has been performed on the solution or not, the hypotheses about the solution should be evaluated in the next stages, starting with an internal validation.

3.4 Internal Validation

The first step on the right side of the V-Model is internal validation, sometimes referred to as *academic validation* or *scientific benefit*. At this stage, the hypotheses about the solution concerning the scientific problem are tested. This means that internal validation is concerned with checking the hypotheses about the qualities of the solution to address the scientific problem. The guiding question for internal validation is:

? What Benefits Does the Solution Offer in Terms of Addressing the Scientific Problem?

Put differently, at this stage, the researcher wants to verify how (or if) the solution improves the state of the art. The benefits to be observed can be derived from the hypotheses defined right after the definition of the solution idea, as mentioned previously. Note, however, that the initial hypotheses may change after the solution is ready, for the very process of implementing the solution can shed new light on the benefits that were not completely understood.

Example (Validating the Automated Context Model Through an Experiment)
The researchers collected contextual data from a real mobile app to apply the implementation of the framework. The tool analyzed the data and automatically generated a context model that revealed, based on the contextual data, how different contexts have influenced a certain user task of interest. Then a controlled experiment was designed to evaluate whether or to which extent the resulting context model (primary factor) can help individuals identify relevant contexts to describe context-aware functionalities. Furthermore, the subjects of the treatment group (i.e., participants who used the context model) also answered a debriefing survey to report on the usability of the context model to support their elicitation task.

Empiricism During Internal Validation

Empiricism is at the core of internal validation. Internal validation is about applying one or more empirical methods to test the hypotheses about the benefits of the implemented solution to address the scientific problem. Empiricism at this stage includes:

- **Defining/refining hypotheses:** After implementing the solution, the researcher is ready to define/review the initial hypotheses and check whether they can be refined or whether new hypotheses should actually be considered.
- **Designing empirical evaluation:** If the evaluation procedure was not designed in the late phase of the solution stage, it has to be defined in the early stages of the internal evaluation.
- **Performing empirical evaluation and reporting on it:** The chosen empirical strategy is carried out, data is collected and analyzed, and the results are documented.

According to our experience, **controlled experiments** have been the most popular empirical strategy used to perform internal validation (comprehensive information about how to conduct controlled experiments in software engineering is organized by Wohlin et al. in [32]; detailed information on how to report controlled experiments is provided by Jedlitschka et al. in [10]). In some cases, internal validation was performed with graduate students of RPTU Kaiserslautern and, in other cases, with professionals at Fraunhofer IESE. In any case, even when professionals participate, the evaluation bears the character of an internal validation because it is not performed in a natural (i.e., industrial) setting where people are experiencing the problem under investigation. However, it is unquestionable that when internal validation is performed with the participation of professionals, they contribute much more practical expertise to the results than students.

Controlled experiments are frequently accompanied by a **survey** organized into two parts: the characterization of the participants (sometimes, this is a separate survey sent in advance to help block participants and ensure a balanced distribution of the subjects in the control and treatment groups) and the collection of post-experiment impressions (debriefing survey). The survey data can shed light on the quantitative analysis that follows the execution of the controlled experiment by providing qualitative and (additional) quantitative data to the researchers. As regards qualitative analysis, a comprehensive introduction on using qualitative methods in ESE is provided by Seaman [29]. **Simulation-based studies** (SBS) can also be used to validate solution proposals. An introduction to SBS can be found in [6]. Finally, in our experience, **case studies** have also been occasionally used to perform internal validation (examples can be found in [14, 26]).

3.5 External Validation

Finally, at the final stage of the V-Model, the researchers perform external validation. This stage is also called *industrial validation* or *practical benefit*. The point now is to check whether practitioners can benefit from the solution in their natural settings to solve their real problems. The guiding question at this stage is:

? To What Extent Is the Solution Beneficial in Practice?

While a controlled experiment in an internal validation may provide evidence that a certain factor (namely, the solution itself or something derived from the solution) can be identified as the cause of measurable improvement in the current situation, this does not necessarily imply that the solution is *practical*, i.e., beneficial to practitioners. External validation aims to strengthen the validation of the solution in a practical setting, which can be supported by empiricism.

Example (Using the Data-Driven Modeling Approach in a Company) A medium-sized car-sharing company joined the research, and the solution was used in a real setting. They have a mobile app that their users use to manage the booking of cars. The agile team responsible for creating and maintaining the app was introduced to the framework and the implementation usage. The team followed the process and used the tool to create context models for different user tasks of interest and then used the models to support creative elicitation sessions within the team in order to identify interesting context-aware functionalities to improve their system. At the end of a period of applying the framework, the researchers interviewed the team to get their impressions of the practical benefit (as well as difficulties) experienced by them.

Empiricism During External Validation

This stage is similar to the previous one (internal validation), which is why the role of empiricism is similar: It is about **defining/refining hypotheses** and **designing**, **performing**, and **reporting on the empirical evaluation**. The fundamental differences are the audience (the evaluation occurs in an industrial environment), the sample size, and, typically, the instrument.

Similar to the practical problem stage, **case studies** and **surveys with practitioners** are the usual empirical strategies for evaluating the solution in a practical setting.

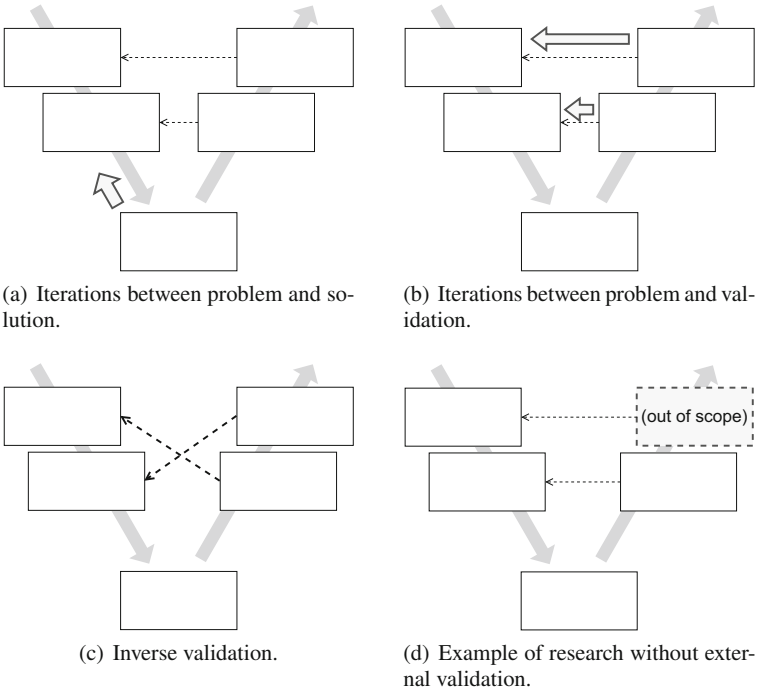
4 Execution Flows and Further Usage Patterns

In the previous section, we presented the most common way in which we have experienced the usage of the V-Model to frame applied research in software engineering using empirical strategies. Although the stages' logical order—practical problem, scientific problem, solution, internal validation, and external validation—suggests a fairly sequential process, it is important to emphasize that the approach does not describe a process but a framework where varying execution flows can occur. Besides that, we have also experienced alternative usage patterns of the V-Model framing approach in several PhD theses done at Fraunhofer IESE and RPTU Kaiserslautern over the years. This section explores those usage patterns.

4.1 Process Execution Flows

While it is possible to follow the stages of the V-Model strictly in sequence, variations and/or iterations are often observed. The most frequent variations are:

- **Multiple iterations between problem(s) and solution definition:** The definition of a solution idea does not begin after the scientific problem is identified. On the contrary, it is natural that potential research ideas, or at least research directions, emerge when a given problem is under investigation. Therefore, as the practical problem and the scientific problem are concretely defined, an early solution idea is refined until it is designed and implemented. See Fig. 3a.
- **Iterations between problems and validations:** During the definition of the practical problem, hypotheses about the future solution can be defined that are expected to be tested, in principle, during external validation. Likewise, when the scientific problem is known, hypotheses about the solution can be identified, and these are expected to be tested, in principle, in the internal validation stage. However, as mentioned in Sects. 3.3 and 3.4, developing the solution may lead to refining the hypotheses and/or introducing new ones. These therefore need to be double-checked in light of the practical and scientific problems. See Fig. 3b.



(a) Iterations between problem and solution.

(b) Iterations between problem and validation.

(c) Inverse validation.

(d) Example of research without external validation.

Fig. 3 Typical variations in the execution flow of the V-Model

- **Inverse validation:** While it has been common usage that the internal (academic) validation tests hypotheses related to the scientific problem and the external (industry) validation tests those related to the practical problem, sometimes, the academic validation (implemented, e.g., through a controlled experiment) tests hypotheses related to the practical problem, and the industry validation (implemented, e.g., through a case study) tests hypotheses of the scientific problem. This can happen for different reasons, such as time and scope constraints of the project where the research has been developed and intrinsic characteristics of the problem under investigation. See Fig. 3c.
- **Either internal *or* external validation:** The reasons mentioned in the previous item also cause another effect in the process execution flow of the V-Model: Depending on the nature of the problem and/or the scope and time constraints of the graduate course, either internal or external validation will be performed (by implementing one or more empirical strategies). One implication is that the test of some declared hypotheses is left out of the scope of some theses. For example, in [7, 25], case studies for external validation were envisioned but not executed within the scope of the theses; in [20], no internal validation was performed but rather a series of case studies in an industrial setting. See Fig. 3d.

4.2 Further Usage Patterns

In most cases, a five-stage V-Model as illustrated in Sect. 3 has been used to frame applied research in empirical software engineering. Nonetheless, alternative patterns have emerged from many theses over the years.

- **Multiple instances of the V-Model:** When PhD research tackles more than one practical problem, each of them with one individual underlying scientific problem, it may make sense to have multiple V-Models to frame the work: one for each practical problem. One example can be found in the PhD thesis of Zhang [33].
- **Research approach and validation approach:** Nearly half of the PhD theses we reviewed used the V-Model to frame the overall research approach or method, usually in the early chapters of the thesis. In some other cases, however, the V-Model was placed in the late chapters related to validation. In yet other cases, the V-Model was used to structure the chapters of the thesis (e.g., [19, 22]).
- **Positioning the hypotheses:** Some theses position the hypotheses on the left side of the V-Model, in connection with the practical and scientific problems. This resembles the original application of the V-Model in the ESE lectures at RPTU Kaiserslautern in the early 2000s. Back then, the left side described the problems with associated hypotheses, which should guide the research; the right side indicated the empirical strategies (often controlled experiments) used to

address the left-side hypotheses.[4] Other theses leverage the structure of the V-Model to visually place their hypotheses on different parts of it (e.g., on the solution stage, on one of the validation stages, or even on arrows linking the validation stages to the problem stages—see one example in [13]).

- **N-stage V-Model:** In some theses, the V-Model does not have five stages. Interesting examples include the usage of a six-stage V-Model as in [2], where the author features three problem stages on the left side and the corresponding three benefit stages on the right side (the bottom stage is not shown, but "solution" is implied there). Similarly, in another thesis [3], the author used a seven-stage V-Model, featuring on the right side the stages "validation: controlled experiment (university context)," "validation: industry case studies," and "roll-out: application in projects."

Despite the variants we found in using the V-Model, the essence of the framing approach remains almost unchanged: In the end, it is all about finding a practical problem, identifying the underlying scientific problem, developing a solution, and validating it internally and externally.

5 Lessons Learned

According to our experience, using the V-Model as a framework for teaching ESE at RPTU Kaiserslautern, the origins of which date back to the early 2000s, has evolved in different ways. In the early days, controlled experiments were the empirical strategy in focus; later, the framework was used to support the structuring of multi-strategy research. Empiricism has been used not only to validate the solution but also to characterize the problem. Figure 4 summarizes the typical empirical strategies applied to each stage of the V-Model.

The various usages that we have experienced over the years also reflect the intended, but perhaps not self-evident, flexibility of the framework. The V-Model helps guide the research process, which is especially beneficial for PhD-level students, where there is frequent uncertainty about what should be done to achieve the final goal. Yet the V-Model should not constrain the research but rather be adapted to the research constraints and goals.

Besides providing guidance along the research process, another feature of the V-Model that has proven beneficial is that the transitions between the stages function as checkpoints, acting as quality gates and monitoring the research progress. While PhD-level research is often solitary work, at these checkpoints, supervisors should contribute their experience to the work by reviewing the results.

[4] A copy of the slide set used in the early 2000s at RPTU Kaiserslautern (formerly University of Kaiserslautern) can be found at https://zenodo.org/doi/10.5281/zenodo.11544897.

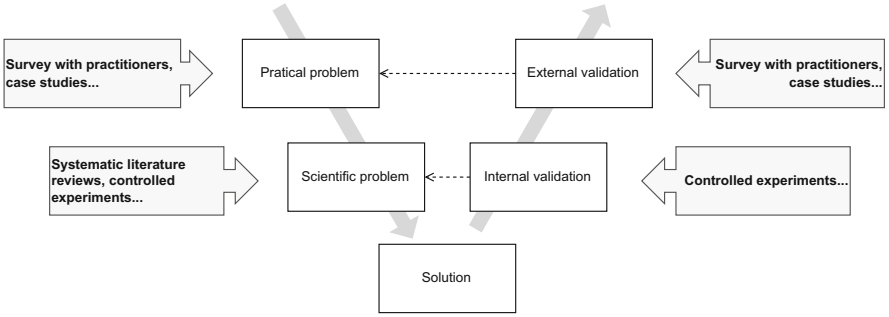


Fig. 4 Summary of the typical empirical strategies used in the stages of the V-Model framework

Feedback from a Survey with Former PhD Students

As we were interested in how former PhD students experienced the V-model during their PhD, we prepared four simple questions to obtain some lessons learned. As mentioned in the introduction, we asked 23 people whose dissertations we had identified as explicit uses of the V-model. We received responses from twelve authors. These have been aggregated into the summary presented in the following.

? "Why did you use the V-model?"

The majority answered that the model was presented and given by the leading PhD supervisor. The model was well established at a certain point and therefore the standard procedure for making progress in one's PhD. As many other PhD students also followed the model and regular discussions and exchange took place among all these students, one learned how others applied the model, saw the advantages, and could reflect on and adapt this for one's own PhD. The model was often considered as a systematic and structured way, and it was considered logical to use the model.

? "How did you use the V-Model? Was it clear for you how the V-Model should or could be used to frame your PhD research?"

The model was sometimes considered as some kind of project plan, i.e., it helped structure the concrete steps of how to proceed and provided guidance on what to do in which order. This means that the students understood how the different steps in the V-model are aligned and what to do in which order. Therefore, it was also considered as some kind of a template, which sometimes was even explicitly covered in the outline of the thesis and the concrete headings. For most, the model became clear (at least after some time) and was then also used implicitly to some extent.

? "What went well in following the V-Model approach?"

One common answer was that the process felt natural and intuitive and was easy to follow and logical. It helped in organizing the work and the written thesis. The good guidance of the model itself together with the discussions helped structure the thesis. Feedback could be given easily this way too. This also meant that the right questions were asked at the right time, implying that a logical order of the steps in the PhD was enforced by following the V-model. Certain phases, such as problem identification or evaluation, were sometimes emphasized, and it was mentioned that the model provides support to address these phases adequately. Finally, the overall alignment of the five steps was rated positively.

? "Were there drawbacks or shortcomings in using the V-Model? What could be improved?"

The most general comment was that some details in the model remain unclear, as there is only little detail available about the individual phases. This was mitigated to a certain extent during the discussions with other PhD students and the leading supervisor. Consequently, the idea was sometimes mentioned that the process should be described in more detail. Some comments address specific steps. For example, it was mentioned that better support in defining the problem could be helpful. Another example was the validation phase where some hints, especially regarding validation with external partners, could improve the evaluation. The evaluation as such might be too small to be generalizable, and the model does not prevent this risk. Also, the relation between the practical problem and the scientific problem was sometimes difficult to draw by some PhD students. Another statement was that the V-model seems to be more accurate for practical PhD theses than for more theoretical theses.

Summarizing the feedback:

- The V-Model has been perceived as an intuitive and useful tool to guide PhD candidates in structuring their research.
- The guidance provided by the approach has helped both the planning of concrete steps and the writing of the theses.
- Students have been able to adapt the V-Model to their own needs.
- The V-Model facilitates feedback giving.
- The approach fits better with applied doctoral research.
- Defining the problem, particularly the scientific problem, is sometimes perceived as a challenging step.

Tip: Identification of the Problem

In our experience, defining the practical problem, especially identifying the underlying scientific problem, is challenging for students. While we acknowledge this difficulty, we see it as a natural part of the applied doctoral process. The identification of a relevant scientific problem, taking a practical problem as a starting point, requires creativity and effort from the candidate to explore the problem space.

6 Further Reading

The principles behind using the V-Model to teach ESE come from the empirical research approach applied at Fraunhofer IESE, which is based on the Quality Improvement Paradigm (QIP). As the QIP has been the reference model adopted by ISERN,[5] its principles are well-known by the ESE community. Therefore, for a process framework for planning, executing, and analyzing individual empirical studies of any type, we refer to the QIP [4], as described in [11]. Figure 5 illustrates how the QIP steps map to the V-Model.[6]

While we contribute a framework for the usage of multiple empirical strategies in applied research in this chapter, empirical strategies themselves have only been mentioned briefly. For a comprehensive introduction to the major empirical

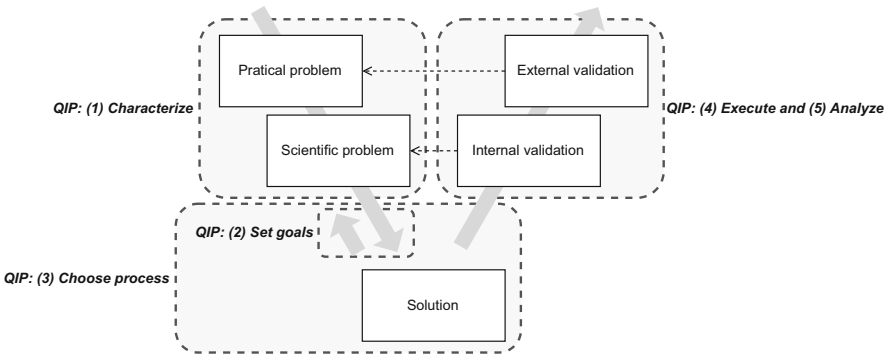


Fig. 5 Mapping between the steps of the QIP and the V-Model research meta-model

[5] ISERN (International Software Engineering Research Network) is the international network of ESE research, aimed at supporting international collaboration on empirical software engineering. https://isern.iese.de/.

[6] The step *Package (6)* refers to the documentation, reporting, and dissemination of the research results. Since the V-Model is used to frame multi-method research, this is a cross-cutting step across the stages. For this reason, it has been omitted in the mapping.

strategies in software engineering, we refer to Wohlin et al. [32]. Advanced aspects of empirical strategies are provided by Shull et al. [30]. For further information about contemporary empirical methods, including case surveys and simulations in empirical software engineering, we refer to Felderer and Travassos [8].

For the operationalization of goals in all types of studies, we refer to the Goal, Question, Metric (GQM) approach [5] and in particular the GQM goal template for formulating goals in a formal way.

The V-Model framework is useful not only in the educational context but also in any applied research setting. In that sense, it is not the only existing framework. For example, an alternative for a multi-method strategy in applied research using ESE is presented by Runeson et al. [27]. In their work, the authors use the Design Science paradigm for framing empirical research, focusing on "articulating and communicating prescriptive software engineering research contributions." Another alternative has been proposed by Gorschek et al. [9]. The authors present a "research approach and technology transfer model" described in a series of steps that sometimes occur in industry (e.g., "problem/issue," "dynamic validation") and other times in academia (e.g., "study state of the art," "validation in academia").

Acknowledgments We acknowledge the direct and indirect contributions of many colleagues and former colleagues at Fraunhofer IESE, the majority of whom were former PhD candidates themselves. They have significantly contributed to building this body of knowledge by applying the V-Model framing approach in various ways over the years.

Appendix

A slide set to support teaching the V-Model framing approach for applied research in ESE that is based on the content of this chapter is available at https://zenodo.org/doi/10.5281/zenodo.11544897 and can be reused by those who may be interested in it.

References

1. Abukwaik, H.: Proactive Support for Conceptual Interoperability Analysis of Software Units. Fraunhofer Verlag (2018)
2. Anastasopoulos, M.: Evolution Control for Software Product Lines: An Automation Layer over Configuration Management. Fraunhofer IRB Verlag (2014)
3. Armbrust, O.: The SCOPE Approach for Scoping Software Processes. Fraunhofer Verlag (2010)
4. Basili, V., Caldiera, G., Rombach, D.: The experience factory. Encyclopedia Software Eng. **1**, 469–476 (1994)
5. Basili, V., Caldiera, G., Rombach, D.: The goal question metric approach. Encyclopedia Software Eng., 528–532 (1994)
6. de França, B.B.N., Ali, N.B.: The role of simulation-based studies in software engineering research. Contemp. Emp. Methods Software Eng., 263–287 (2020)
7. Falcão, R.: Data-Driven Context Modeling for the Elicitation of Context-Aware Functionalities. Fraunhofer Verlag (2023)

8. Felderer, M., Travassos, G.H.: Contemporary Empirical Methods in Software Engineering. Springer (2020)
9. Gorschek, T., Garre, P., Larsson, S., Wohlin, C.: A model for technology transfer in practice. IEEE Software **23**(6), 88–95 (2006)
10. Jedlitschka, A., Ciolkowski, M., Pfahl, D., Shull, F., Singer, J., Sjøberg, D.: Reporting controlled experiments in software engineering. In: Shull, F., Singer, J., Sjøberg, DI (eds.) 2005 International Symposium on Empirical Software Engineering, Noosa Heads, QLD, Australia (2007)
11. Jedlitschka, A., Guzmán, L., Jung, J., Lampasona, C., Steinbach, S.: Empirical practice in software engineering. Perspectives on the Future of Software Engineering: Essays in Honor of Dieter Rombach, pp. 217–233 (2013)
12. Jedlitschka, A., Pfahl, D.: Reporting guidelines for controlled experiments in software engineering. In: 2005 International Symposium on Empirical Software Engineering, pp. 10–pp. IEEE (2005)
13. John, I.: Pattern-Based Documentation Analysis for Software Product Lines. Fraunhofer Verlag (2010)
14. Jung, C.: Context-Aware Security. Fraunhofer Verlag (2019)
15. Kitchenham, B., Charters, S., et al.: Guidelines for Performing Systematic Literature Reviews in Software Engineering (2007). Technical Report
16. Kitchenham, B., Pfleeger, S.L.: Principles of survey research – parts 1–6. ACM SIGSOFT Software Engineering Notes, pp. 26–28 (2001–2003)
17. Knodel, J.: Sustainable Structures in Software Implementations by Live Compliance Checking. Fraunhofer Verlag (2011)
18. Mathur, S., Malik, S.: Advancements in the v-model. Int. J. Comput. Appl. **1**(12), 29–34 (2010)
19. Menzel, I.: Optimizing the Completeness of Textual Requirements Documents in Practice. Fraunhofer Verlag (2012)
20. Müller, M.: Analyzing Software Quality Assurance Strategies Through Simulation. Fraunhofer IRB, Stuttgart (2007)
21. Naab, M.: Enhancing Architecture Design Methods for Improved Flexibility in Long-Living Information Systems. Fraunhofer Verlag (2012)
22. Ocampo, A.: The REMIS Approach to Rationale-Based Support for Process Model Evolution. Fraunhofer IRB Verlag (2009)
23. Petersen, K., Feldt, R., Mujtaba, S., Mattsson, M.: Systematic mapping studies in software engineering. In: 12th International Conference on Evaluation and Assessment in Software Engineering (EASE) 12, pp. 1–10 (2008)
24. Rook, P.: Controlling software projects. Software Eng. J. **1**(1), 7–16 (1986)
25. Rost, D.: Task-Specific Architecture Documentation for Developers. Fraunhofer Verlag (2019)
26. Rudolph, M.: Generation of Usable Policy Administration Points for Security and Privacy. Fraunhofer Verlag (2020)
27. Runeson, P., Engström, E., Storey, M.-A.: The design science paradigm as a frame for empirical software engineering. Contemp. Empir. Methods Software Eng., 127–147 (2020)
28. Runeson, P., Host, M., Rainer, A., Regnell, B.: Case Study Research in Software Engineering: Guidelines and examples. Wiley (2012)
29. Seaman, C.B.: Qualitative methods in empirical studies of software engineering. IEEE Trans. Software Eng. **25**(4), 557–572 (1999)
30. Shull, F., Singer, J., Sjøberg, D.I.: Guide to Advanced Empirical Software Engineering. Springer (2007)
31. Trendowicz, A.: Software Effort Estimation with Well-Founded Causal Models. Fraunhofer-IRB-Verlag (2008)
32. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering. Springer Science & Business Media (2012)
33. Zhang, B.: VITAL-Reengineering Variability Specifications and Realizations in Software Product Lines. Fraunhofer Verlag (2015)

Sustainability Competencies in Research Strategy for Software Engineering—Personal Experience and Best Practices

Birgit Penzenstadler

Abstract This chapter provides an experience report with a particular focus on lessons learned emerging from onboarding BSc, MSc, and PhD students for empirical research projects and supervising them to design their own.

With the objectives to (1) prepare students as agents of change that master the sustainability competencies and (2) teach students good research methods and proper empirical research design, I provide a list of supervision and onboarding scenarios that have taught me what works and what does not work so well and how to improve. As guidance, I focus on exercises that can be used in teaching at all levels and supervision scenarios that are more useful for PhD student mentoring.

The chapter discusses aspects related to didactics and pedagogy, shows how to tie the ABC of software engineering research in synergistically with the sustainability competencies framework and positions the work within the leverage points for sustainability. It concludes with recommendations and takeaways for best practices as well as teaching materials and reflections on ethics.

1 Introduction

Sustainability science includes a notion of resilience as adaptation over time instead of understanding sustainability as maintaining a status quo. Software engineering has been shown to have an impact on this development [15]. While many initiatives to transition toward sustainability via software-intensive systems development have led to improvements of sustainable development goal targets,[1] the underlying computational thinking in combination with disciplinary silos creates shortcomings.

[1] https://sdgs.un.org/goals.

B. Penzenstadler (✉)
Chalmers University of Technology and University of Gothenburg, Gothenburg, Sweden

Lappeenranta Lahti University, Lappeenranta, Finland
e-mail: birgitp@chalmers.se

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2024

D. Mendez et al. (eds.), *Handbook on Teaching Empirical Software Engineering*,
https://doi.org/10.1007/978-3-031-71769-7_10

Software systems based on computational thinking are effective at solving specific challenges—in contrast, sustainability challenges require highly transdisciplinary approaches. In our highly effective engineering disciplines, we attempt to frame and solve many of the challenges we like to call problems, but we are not necessarily well trained in these skills. Consequently, we can fall short in seeing the holistic picture, the side effects, the disadvantaged populations affected by our "solutions," and the discounting of the future as well as other life forms in domains such as health, education, and communal living. To mitigate this conundrum, we need agents of change that are trained in transdisciplinary sustainability.

There are a few topics that we get to train students in orthogonal to whichever specific discipline or topic they study or do research in. These topics include ethics, diversity, inclusion, equity and accessibility, and sustainability. If we humans don't manage to change the trajectory of our society from consumerism toward stewardship, then we may push the planet so far beyond its boundaries for safe operation that we get to deal with consequences of an unstable or even uninhabitable environment [16]. Rockström and Sachs believe for the continuation of society past 2015 "that the Business-As-Usual path is the most likely scenario and that it will lead to a highly unequal world that is also unstable and often violent" [16, p. 6]. Change is needed, according to [16], in energy transition, food security, urban sustainability, population transformation, biodiversity management, and private and public governance. The responsibility for change lies in all of us and more so with the privileged part of society that, as researchers, we tend to be part of—having sufficient resources to survive comfortably and hence dedicate ourselves to helping others. Since most of us, researchers, care that our work has a positive impact on the actual world, it seems a natural tendency to include explicit thoughts and education on sustainability through supervising research design.

There are several ways to think about sustainability in the context of supervising research design:

1. Sustainability (of a human society) as an central objective to contribute to via a chosen research topic
2. Sustainability as a secondary objective within any chosen research topic
3. Sustainability mindset as contributing to any process or service
4. Sustainability mindset in choosing one's research design

Why Read This Chapter In this chapter, you learn how to (1) prepare students as agents of change that master the sustainability competencies and (2) teach students good research methods and proper empirical research design that adheres to sustainability principles. This chapter provides:

1. A list of supervision and onboarding scenarios that show what works and what does not work so well and how to improve and deepen over time.
2. A discussion of aspects related to pedagogy and that show how to tie the ABC of software engineering research [17] in synergistically with the sustainability competencies framework [2] and position the work within the leverage points for sustainability [1].

In the following, Sect. 2 describes the background and foundations for this chapter. Section 3 proposes the integration of sustainability competences into software engineering research design, and Sect. 4 discusses ethical aspects.

2 Background and Related Work

There are three main works that this chapter is based upon: (i) the key competencies for sustainability, (ii) the ABC of software engineering research, and (iii) the leverage points.

2.1 Key Competencies for Sustainability

Brundiers et al. [2] conduct a Delphi study with 14 international experts in sustainability education on the framework of key competencies in sustainability by Wiek et al. [22], the most frequently cited framework to that date, and propose a slightly refined version: The established key sustainability competencies are systems thinking competency, strategic competency, values-thinking (or interpersonal) competency, future (or anticipatory) competency, and implementation (or problem-solving, normative) competency [2].

The learning objectives for each of those are as follows [2, p. 22]:

- **Futures-thinking competency**: to be able to iterate and continuously refine one's own futures thinking (visions, scenarios, etc.), in productive and explicit tension to the status quo, recognize "implicitly held (and largely unrecognized) assumptions about how society works" and how they influence the status quo, and critically reflect on how they might influence futures thinking.
- **Values-thinking competency**: to be able to differentiate between intrinsic and extrinsic values in the social and natural world; to recognize normalized oppressive structures; to identify and clarify one's own values; to explain how values are contextually, culturally, and historically reinforced; to critically evaluate how particular stated values align with agreed-upon sustainability values; and to differentiate between espoused values and practiced values.
- **Strategic-thinking competency**: to be able to recognize the historical roots and embedded resilience of deliberate and unintended unsustainability and the barriers to change; to creatively plan innovative experiments to test strategies.
- **Interpersonal competency**: to be able to apply the concepts and methods of each competency not merely as "technical skills" but in ways that truly engage and motivate diverse stakeholders and to empathically work with collaborators' and citizens' different ways of knowing and communication.
- **Integrated problem-solving competency**: to be able to combine and integrate steps of the sustainability problem-solving process or competencies while draw-

ing on pertinent disciplinary, interdisciplinary, transdisciplinary, and other ways of knowing.

This framework of key competencies is used by the author to adapt software engineering research education toward a more sustainability-oriented mindset.

2.2 The ABC of Software Engineering Research

Stol and Fitzgerald open their seminal paper on the strategies of Software Engineering research with the statement "The choice of research strategy is not "good" or "bad" a priori but is very much dependent on the setting and goal of the research" [17, p. 1]. They choose the term research strategy to denote a category of research methods characterized by two dimensions: (a) the level of obtrusiveness of the research in a given setting and (b) the extent to which the findings are generalizable to other settings. They describe three concerns to frame their approach, Actors, Behavior, and Context (hence ABC), and propose eight archetypal research strategies: Field Study, Field Experiment, Experimental Simulation, Laboratory Experiment, Judgement Study, Sample Studies, Formal Theory, and Computer Simulation. The author of this chapter uses the research strategies to educate junior researchers.

2.3 Leverage Points

As software engineers, we are responsible for the long-term consequences of the systems we design—including impacts on the wider environmental and social sustainability. However, the field lacks analytical tools for understanding these potential impacts while designing a system or for identifying opportunities for using software to bring about broader societal transformations. The concept of leverage points as known from system dynamics is used to make sustainability issues more tangible in software systems design [15].

The twelve leverage points (LPs) to intervene in a system were proposed by Donella Meadows [14], a scientist and system analyst who studied environmental limits to economic growth. They are places within a complex systems, e.g., a corporation, an economy, a living body, a city, or an ecosystem, where a small shift in one thing can produce big changes in others. They can be grouped into four categories of system characteristics of parameters, feedbacks, design, and intent, where changing them creates effects in increasing order of magnitude. The detailed explanation for them is given in Table 1, and an illustrated overview is given in Fig. 1.

Table 1 Leverage points as defined by Meadows [14]

Group	LP	Description
Parameters	LP12	Constants, parameters, and numbers. Tweaking parameters allows change to the intensity of the flows in systems but rarely alters the underlying dynamics
	LP11	The sizes of buffers and other stabilizing stocks, relative to their flows. Stabilize a system by adjusting the capacity of its buffers, and make it more efficient by optimizing the flow
	LP10	The structure of material stocks and flows (such as transportation networks and population age structures). Physical structure is crucial in a system but often hard to change; therefore, the leverage point is in proper initial design
Feedbacks	LP 9	The lengths of delays, relative to the rate of system change. A system cannot respond to short-term changes when it has long-term delays
	LP 8	The strength of balancing feedback loops, relative to the impacts they respond to. Balancing feedback loops helps systems self-correct by monitoring and adjusting according to the system goal
	LP 7	The gain around reinforcing feedback loops. Reinforcing feedback loops can be sources of system instability or mechanisms to amplify desired change, so adjusting their strength affects how the system responds to change
Design	LP 6	The structure of information flows. This can create a new feedback loop that was not there before. Altering the structure of information flows enables more agency by users
	LP 5	The rules of the system, including incentives, punishments, and constraints. Social rules include constitutions, laws, standards, policies, and incentives. Changing the rules of a system can change the behavior of the society under them
	LP 4	The power to add, change, evolve, or self-organize system structure. In biology, this is called evolution; in society, we call it empowerment. In systems terms, it is called self-organization, the strongest form of system resilience
Intent	LP 3	The goals of the system. Changing the goal of a system is a powerful strategy to effect change but can be hard to achieve
	LP 2	The mindset or paradigm out of which the system arises. Paradigms are a shared set of deep beliefs about how the world works. They are the hardest to change in a system, as society will fiercely resist any challenges to its paradigms
	LP 1	The power to transcend paradigms. This final and most effective LP is about being unattached to existing paradigms; there is no certainty in any particular worldview

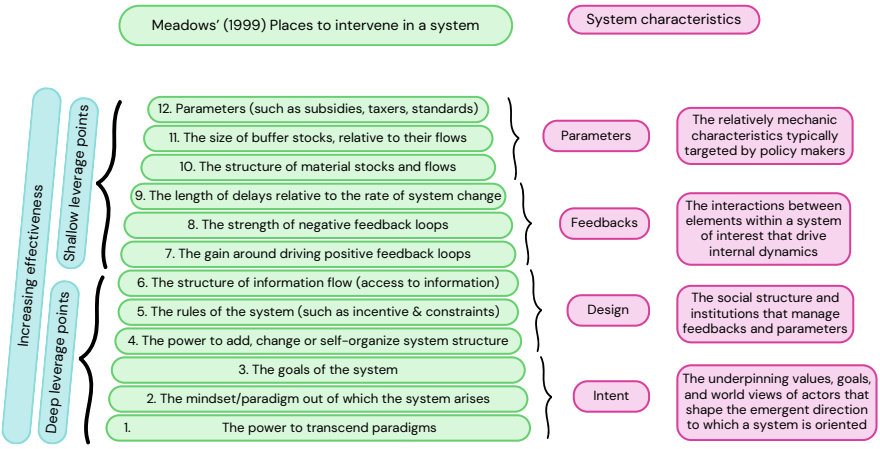


Fig. 1 Leverage points illustrated according to their effectiveness, redrawn after [14]

2.3.1 Leverage Points for Sustainability

Since this chapter is about sustainability, let's also mention Abson et al. [1], who discuss the application of leverage points in sustainability transformation and find three realms of deep leverage:

1. Re-structure: change, stability, and learning in institutions
2. Re-connect: targeting interactions between people and nature
3. Re-think: how knowledge is produced and used

Dave Abson, the first author, gave a keynote on using leverage points for sustainability transformations the same year I had published the article about using them in software engineering. Our engaged conversation at the LIMITS conference that year with the entire community confirmed how useful they are in bridging thinking across different disciplines.

2.3.2 LP Application Within Software Engineering

Leverage points can help software engineers address sustainability challenges by offering insights on possible transformation mechanisms or ways to find alternatives [15]. In increasing order of the likely magnitude of their effect, they can support:

1. Changing the metabolic structure (LP12, LP11, and LP10)
2. Changing the feedback loops (LP9, LP8, and LP7)
3. Transformational change and self-adaptation (LP6, LP5, and LP4)
4. Changing the intent of the system and stakeholders (LP3, LP2, and LP1)

While LPs will not tell us exactly how to act on sustainability challenges, they provide an analysis tool to help practitioners identify elements that can bring about effective change at different levels, for a (software) system and the wider system it resides in. In the mentioned IEEE Software article [15], I provide an example of the UK public transport system. I use these leverage points in several spaces within teaching and mentoring. I have spoken to my bachelor students about them, and they can follow the transport example quite well. Same with the MSc students, who I try to expose to a bit more systems thinking in general, for example, using exercises from The Systems Thinking Playbook [18].

In the following, I explain how I teaching these concepts both in the classroom settings and in PhD student supervision.

3 Sustainability Competencies in Research Strategies for Software Engineering

Within transdisciplinary education, as suitable for research that serves sustainability science, it is necessary to understand how the key sustainability competencies can be discussed within the research strategies we consider useful for empirical software engineering. In this section, we look at how the key competencies for sustainability [2] can support a researcher in finding the adequate research strategies [17] within a software engineering topic and how these strategies can touch upon the leverage points for sustainability [1, 15]. The big picture overview is illustrated in Fig. 2.

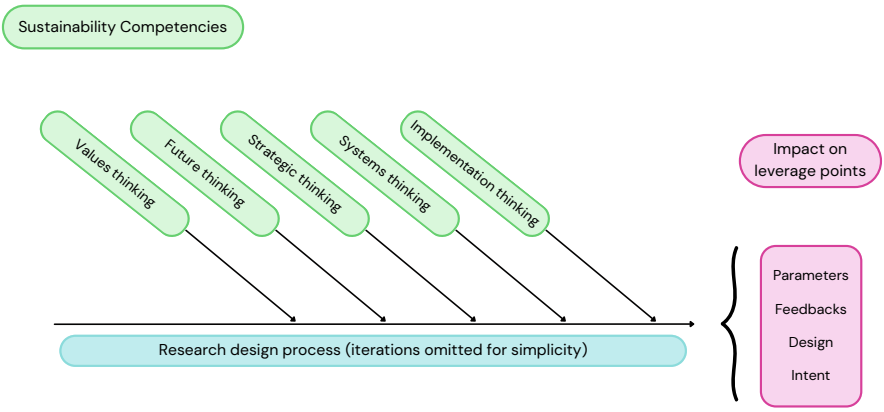


Fig. 2 Big picture of sustainability competencies' influence on research design and the insights on leverage points

3.1 Preparing Agents of Change: Teaching the Competencies

The key competencies of sustainability are "not naturally developed in teaching–learning settings" [2, p. 26] but require targeted and ongoing efforts to learn about competencies and through working with each competency's set of concepts, methods, and skills. Consequently, they need to be taught within the first year of a PhD supervision, which I call "Preparing Agents of Change."

3.1.1 Systems Thinking Competency

There are plenty of good resources available to learn and teach systems thinking. One of my personal favorites is The Systems Thinking Playbook [18], a collection of games to be played individually or in groups that teach how to think in and about systems in a structured way while keeping the context entertaining. Further games are available [10]. A bit more tailored toward sustainability in science education are the chemistry-oriented examples in Blatti et al. [3] that also bring in the UN Sustainable Development Goals and the planetary boundaries [16].

> Exercise

The first step of developing systems thinking competency is to familiarize oneself with the general ideas. While BSc and MSc students can generally follow the explanations of leverage points reasonably well on a cognitive level, it does require a lot of abstract thinking. Hence, I like to get them to play in order to engage other parts of their brain to get a more experiential understanding of the systems thinking concepts. The Systems Thinking Playbook [18] has a great variety of such exercises. One of my favorites is the *Fishing Game*, where teams get to decide how much they choose to fish out of a common lake over a year, given a certain replenishment rate. If one or several teams become a little greedy, everybody ends up with an empty lake after only three rounds.

! Supervision

With my PhD students, I like to go deeper. We walk through what the *system* is in their case, e.g., the scope under consideration for their PhD thesis, and then what the leverage points are within that system. I recommend to schedule a 2-hour slot for such an exploration so that the student has enough time to think it through, or it can be started in one supervision session and then completed as draft at home and discussed in a follow-up session.

3.1.2 Strategic Competency

First, to recognize the historical roots and embedded resilience of deliberate and unintended unsustainability and the barriers to change, a student gets to engage with some of the literature around concepts like rebound effects and planned obsolescence [12]. Second, to creatively plan innovative experiments to test strategies. This means understanding the strategic advantages and disadvantages certain research strategies [17] bring with them in terms of what they enable us to understand and see and where their boundaries of insight are.

> Exercise

The first step of developing strategic competency is to understand that trajectory of our history of (un)sustainability. One of my go-to resources is the book chapter by Lorenz Hilty, who founded the field of sustainability in informatics around 1990. The chapter gives a good overview of what aspects of sustainability we get to think about and introduces concepts like energy efficiency, rebound effects, obsolescence, and substitutability [12]. We discuss this in class afterward to digest the reading and take application examples the students bring in from their experience or curiosity.

3.1.3 Values-Thinking Competency

My favorite research on emotions is conducted by Brené Brown, who over the last 20 years has published a number of books looking at traits supporting identity, belonging, and leadership—research summarized in the Atlas of the Heart [7], which gives an overview of the entire spectrum of human emotions. Such a foundation is valuable to start exploring our own experiences, especially when intentions don't match up with outcomes or first results. In my experience, going through a PhD is a tremendous growth experience, and all growth comes with struggle. As supervisors, we get to choose how personal we get in supporting a student beyond the research content. My personal conviction is that the outcome of a PhD is the researcher more over the thesis, and that includes conversations on what informs our research, what impacts we wish to create in the world, and where our boundaries are. Forming one's character is helped significantly by getting clear on what we value.



Fig. 3 Collection of values to select from or add one's own, inspired by [6]

> Exercise

The first step of developing values thinking competency is to become crystal clear on our own values. I have conducted this exercise in numerous educational settings, including university classroom settings in software engineering, university summer courses on sustainability, public yoga classes, and large online classes on mindfulness and resilience. Depending on the framing of the class setting, I explain that our actions are rooted in our values, whether we are conscious of them or not. Therefore, whether we are designing a system or having a conversation with someone, it is important to know which inner values influence our choices so we can either choose them consciously or reflect upon them and change if we wish. I display or hand out Fig. 3 and give students a few minutes to reflect. If someone feels stuck, I guide them to remember recent situations and what values they can infer from the choices they made in those situations. Usually, the first iteration ends up with many chosen values, so then I ask them to narrow this down to no more than three that they wish to focus on over the period of a month. I invite them to do **reflective journaling**, ideally every night, otherwise in a weekly review, of how they lived into those values and whether that made any difference to them.

3.1.4 Future Competency

In order to be able to iterate and continuously refine one's own futures thinking (visions, scenarios, etc.), one must develop an initial version. This can be achieved

with the help of inputs from storytelling [9], speculative design,[2] and design fiction [8, 19].

Such visions and scenarios can evolve in productive and explicit tension to the status quo. Noticing and understanding such tensions requires recognizing implicitly held assumptions about “how society works" and how such beliefs influence the status quo. Based on that, a student may critically reflect how the identified beliefs might influence futures thinking.

> Exercise

The first step of developing future competency is to become familiar with one or two techniques of creating future scenarios. In an informal way, since we may not always have time to introduce more techniques, I ask students to spend 5 minutes talking to their neighbor about what the future might look like if the system we are discussing becomes wildly successful and has a billion users. After those 5 minutes, one group gets interviewed about certain consequences such success might bring, and we speak about what constraints should be in place to enable likely benefits or mitigate risks that come along.

3.1.5 Implementation Competency

We develop implementation competency by practicing a process that roughly iterates over the phases of ideate, plan, implement, analyze, and reflect. Keeping a research journal has proven an essential part of my personal research practice, and I encourage all students I supervise to keep their own research journal as a reflective assessment method. While I prefer handwritten notes to take the reflection process deeper, others might find digital notes more convenient.

A crucial thought I strive to ingrain in my students from the start is framing of failure as a learning experience.

> Application

The first step of developing implementation competency is to master a basic version of project management and iterative development. So whatever course I teach, deliverables get feedback on a first submission and then a chance to be improved. Team assignments have team contracts that specify roles as well as agreements on communication and collegiality.

[2] https://speculative.hr/en/introduction-to-speculative-design-practice/.

Table 2 Sustainability competencies and their impact on research design

Sustainability competency	Impact on research design
Values-thinking	Select general research topic "I care about X"
Futures thinking	Choose what to focus on "I want to look at Y to contribute"
Strategic thinking	Develop research objectives (i) Understand Y in relation to X (ii) Improve Y in relation to X (iii) Increase/disseminate impact
Systems thinking	(i) Develop research questions for each (ii) Select adequate research strategies
Implementation thinking	(i) Check context, variables, … (ii) Details of research design

3.2 Applying the Competencies: Designing Research

In Table 2, we see the overview of how the sustainability competencies can guide the development of a research strategy.

3.2.1 Values-Thinking Competency

Based on the discovery work in getting familiar with values, a student can choose the area they wish to focus on for their research. This is denoted in Table 2 as "I care about X," for example, "I care about the well-being of IT workers."

! Supervision

I use the same exercise for values with Table 3 in one-on-one supervision if a student is looking for direction or guidance. This often happens the first time they run out of time and/or energy during their PhD, and then they benefit from support to get back into alignment with their values, into the sweet spot (see Fig. 4).

3.2.2 Future Competency

The basic skills on one or more future scenario techniques can help in this step to develop a scenario of what a student wants to focus on. For a given future scenario and topic they care about, they choose a specific aspect as their main center of attention, for example, "I want to focus on individual resilience of IT workers."



Fig. 4 The alignment zone, also called sweet spot or flow

Supervision

During the stage of a PhD where a topic is established, I ask the student about a vision of what success might look like for them—specifically, if they can envision a scenario where they made the world a slightly better place through the research they contributed. For example, if a student is passionate about education on diversity, maybe they envision a set of teaching templates that make it easier for educators to make their existing classroom materials more diverse. Think GenderMag http://gendermag.org/ for all sorts of diversity. That does not mean that every student will be able to create such a clear vision since research often develops over time, but playing with scenarios can be tremendously helpful also for establishing scope boundaries.

3.2.3 Strategic Competency

Based on the choice made for the research focus, the student can now start thinking about a strategy to contribute to the topic. This usually requires several steps that have different research strategies and that contribute a larger nugget of research, for example, to first get a better understanding of a circumstance, for example, via surveys, interviews, or observations, and then to develop a positive improvement, for example, via experiments, simulations, or interventions. For the focus on individual resilience of IT workers, the understanding could be an interview study with IT workers to explore the factors contributing to their well-being and resilience, and then a second stage could be about developing a training to increase well-being and resilience.

Table 3 attempts to map the impact by the key competencies for sustainability for each research strategy.

Table 3 Research strategy and impact by key competency for sustainability

Research strategy	Systems t.	Strategic	Values-t.	Future	Implement
Field study	X	–	–	–	–
Field experiment	X	–	–	–	–
Experimental simulation	–	–	–	–	X
Laboratory experiment	–	–	–	–	X
Judgement study	–	–	–	X	–
Sample studies	–	–	–	X	–
Formal theory	–	X	–	–	–
Computer simulation	–	–	–	–	X

Supervision

I have a conversation with the student to hear about their understanding of the content of the earlier mentioned Hilty introduction chapter on sustainability within and via IT [12] and how they think it applies to their research. We speak about what obsolescence hardware is relevant in their project scope, if efficiency or substitutability is an objective, and what kind of rebound effects may potentially cause as a side effect.

Second, we discuss the strategic advantages and disadvantages the different research strategies [17] would bring with them in the project under consideration. For example, if we conduct a field study, we learn a lot about the actual real-world context, but we are losing comparability when trying to establish a benchmark.

3.2.4 Systems Thinking Competency

Within a chosen research strategy, we get to reflect on what we are affecting within a system, so it is good to model the system and leverage points. We break down which parts of it are in scope for a particular research study and which ones are out of scope. The discussion of scope is called boundary critique [20].[3]

To help those working with leverage points to position their work, understand how it fits to others, and remain sensitive to the impacts of the normative framings they introduce, Leventon et al. [13] offer the following reflection questions:

1. What is the system of focus, and what are its properties (paradigm, design, processes, and materials)?
2. What are the problem framings and norms that underpin this system framing?

[3] https://wulrich.com/boundary_critique.html.

3. What systems is the focal system nested within (multi-scale systems) or connected to (different system framings)?

Working to transform systems therefore requires that we are able to navigate the impact of interventions and their consequences (intended and unintended) within and between systems, directly and indirectly. Indeed, dancing with systems means that we need to be able to respond to changing situations and embrace emerging system dynamics. To deal with this complexity, Leventon et al. [13] offer the following guiding questions for interrogating interventions:

4. Which system properties (paradigm, design, processes, and materials) does the intervention target, and in which focal system?
5. What properties are impacted over time or space or via indirect impacts?
6. How does that intervention influence and work in connected or nested systems?

The research practice framed by this contribution is one of transdisciplinarity; this may (or may not) be a transdisciplinary research design, based on co-creation, but it is clearly an internal, personal process. It provides space to reflect on the questions 1–6 outlined above. However, it also goes further and requires that we reflect on how we engage with a system and how we can converge around the creation and understanding of boundary objects. It facilitates dancing with systems by encouraging us to embrace uncertainty, listen to what the system tells us, stay as a learner of the system, and question our role in the system (see Meadows [14]). Leventon et al. [13] therefore add a further three guiding questions to shape a leverage points research practice:

7. Where am I in the system?
8. What are the boundary objects within this system?
9. How do I act, and what normative framings do I add to this system?

Exercise

I personally am a fan of integrating aspects like sustainability, inclusion, and ethics into any course I teach, and for each assignment, for example, in software design, I add a task with a reflective component. Take the above three sets of questions by Leventon et al. [13] for a case study with the students that they have already been working on for an in-class discussion session.

3.2.5 Implementation Competency

To deepen the implementation competency, we reflect on what worked well in a first iteration of a project and then what needs to be improved in terms of (i) research method skills, (ii) administrative or organizational skills, and (iii) personal and

interpersonal skills. All of these are useful to reflect on in an individual review and then later on in conversation with a supervisor or mentor.

Supervision

During the scoping phase of the research, I let a student develop an online whiteboard that provides a big picture illustration of the research area and scope of the project. This big picture is updated in iterations and may change significantly over the course of the PhD. With that brainstorming canvas on what scoping would serve for the project under consideration, we discuss the research methods [17] that we consider applying to understand the implementation constraints and challenges they bring and how to deal with them. Then the student formulates a plan for carrying out the research design.

To make it easier for the readership of this book and your students, we provide a summary, Sustainability Research Canvas (Fig. 5), to help sketch all suggested reflections in this section in one page that can be iterated over time.

3.3 Becoming a Sage: Reflecting on Research

One thing that, in my experience, we often do not take sufficient time for after completing a study is a reflective review. What did we learn from the study not only in terms of outcomes but also about our research design choices, and how that might inform our research going forward? Structuring these thoughts according to the sustainability competencies would be one iteration of solidifying them. In this section, we look back at Table 2 and ask questions to take the learning deeper.

3.3.1 Systems Thinking Competency

Reflecting on how systems thinking was applied in a piece of research, one can ask: Which assumptions did we (maybe accidentally) make about the system or its environment that turned out to be wrong? What influence does the particular chosen research strategy have on the leverage points of the system under consideration? The general conceivable impacts are listed in Table 4, and that table should be instantiated and detailed for a particular study under consideration. For an easier overview, we illustrate the content in Fig. 6.



Fig. 5 Sustainability research canvas template

Table 4 Research strategy and potential impact/insight on leverage points group

Research strategy	Parameters	Feedbacks	Design	Intent
Field study	X	–	–	–
Field experiment	X	–	–	–
Experimental simulation	X	X	–	–
Laboratory experiment	X	X	–	–
Judgement study	X	X	X	–
Sample studies	X	X	X	–
Formal theory	X	X	X	X
Computer simulation	X	X	X	X

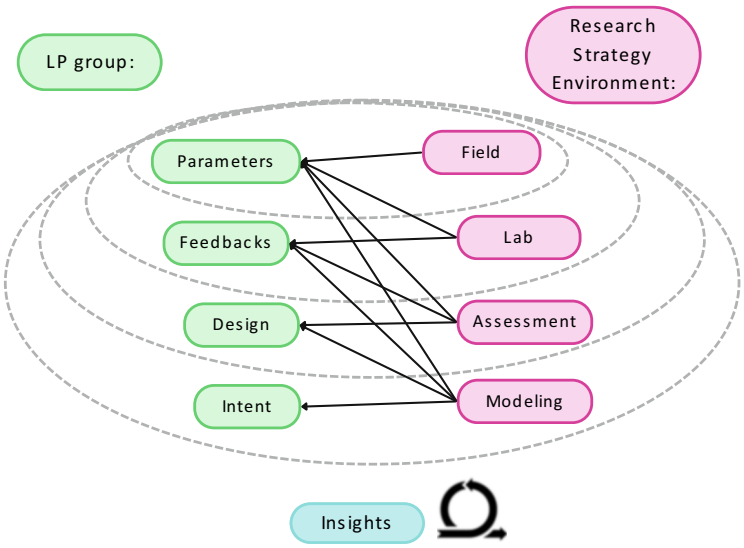


Fig. 6 Likely insights on leverage point groups within each research strategy environment

Supervision

Reflection and a conversation on impacts can be facilitated with guiding questions that ask: (1) What is the effect of Research Strategy X on the leverage points we identified earlier for the system under consideration? (2) How do the results impact that?

3.3.2 Strategic Competency

What in our strategy worked out in terms of bringing us closer to the impact we would like to create? What did not work out and needs a course correction?

To reflect on strategic competency, we look at the general literature to develop leadership. Goldmann's study [11], using phenomenological interviewing technique with CEOs, identified ten experiences that contributed to the development of a person's ability to think strategically, and those experiences represented four levels of interaction: personal, interpersonal, organizational, and external.

- Personal: family upbringing/education, general work experiences, becoming a leader
- Interpersonal: being mentored, being challenged by a key colleague
- Organizational: monitoring results/benchmarking, doing strategic planning, spearheading a major growth initiative
- External: dealing with a threat to organizational survival, vicarious experiences

The three developmental patterns that Goldmann observed were the development of understanding, the practice of rational planning, and the completion of a hierarchy of challenges.

> Exercise

As a reflection step for a student, one could suggest to review the above ten types of experiences and detail them with personal narratives. Such reflections are a useful practice for storytelling and mentoring.

3.3.3 Values-Thinking Competency

In ethnographic research, positionality statements have been proposed long ago as a necessary framing of a research context. They are meant to address genuine concerns about the limits of knowledge production. There are scholars who argue that this should be taken care of by the integrity of the collective process. Both may be true, and it is a useful exercise to reflect upon one's results in light of what the motivation was and whether and how that may have influenced any results.

! Supervision

Guiding questions to ask here are: (1) Did the study actually do what it was supposed to do? (2) Did it support the action I wanted to take, and am I (still) in alignment with the research?

3.3.4 Future Competency

This competency reflection helps iterate the chosen focus and extrapolation of first indicators. More formally, this could help in developing better hypotheses in the future.

Supervision

Reflection questions that may serve here are: (1) What unexpected results did we get, and how do we make sense of them? (2) What does that tell us about the way we made assumptions? (3) How does this inform how we want to make assumptions in the future?

3.3.5 Implementation Competency

Over the implementation phases, I encourage my research students to write debriefing notes for themselves after every milestone on what worked well and what went differently from what they expected. Such insight may sometimes serve as seed for methodology papers that review lessons learned. A template can be found in the resource repository.[4]

Supervision

The research diary should have a weekly review page that asks: (1) What did I learn this week? (2) What went well and what did not? (3) What in the research design may need to be adjusted in a next step?

4 Discussion of Ethical Aspects

We see teaching sustainability in order to empower students as agents of change as a stance that already strongly advocates for individual reflection on values and the impact it creates on the world. This section discusses three aspects tied to related work that could offer further guidance to take the topic deeper.

[4] https://zenodo.org/doi/10.5281/zenodo.11544897.

4.1 Contribution to Emergentist Education

Barrineau et al. [4] discuss the question of how we conceptualize and work toward an education that enables people to anticipate and contest visions of futures and presents in order to "fight for a viable future for all." They identified three themes of common interest from the experiences of their multidisciplinary contexts to explore (1) disciplines and institutional cultures, (2) anticipatory emotions, and (3) the paradox of sustainability as emergent through radical futurity. The authors conclude that these areas point toward opportunities for rethinking norms in higher education, including disciplines, emotions, and conceptions of sustainability.

We see the chapter at hand as a contribution toward emergentist education that enables students as agents of change.

4.2 Transformational Sustainability Research

Transformational sustainability research develops evidence-supported solution options for sustainability problems. According to Wiek and Lang [21], sustainability science can roughly be differentiated into two distinct research streams—a "descriptive-analytical" and a "transformational" one. Their proposal for transformational sustainability research includes capacity for (1) complex problem-handling, (2) transition management and governance, (3) backcasting, and (4) integrated planning research (transdisciplinary case studies).

In comparison to the ABC of software engineering research discussed earlier, the transformational sustainability approach takes a stronger approach of implementing research as change and transition toward a more sustainable human lifestyle.

4.3 GreenComp: The European Sustainability Competence Framework

The European Union defined a set of sustainability competencies called "GreenComp" in 2022 that consists of 12 competences organized into four areas [5]:

1. Embodying sustainability values, including the competences: valuing sustainability, supporting fairness, and promoting nature
2. Embracing complexity in sustainability, including the competences: systems thinking, critical thinking, and problem framing
3. Envisioning sustainable futures, including the competences: futures literacy, adaptability, and exploratory thinking
4. Acting for sustainability, including the competences: political agency, collective action, and individual initiative

All competences are illustrated with learning objectives on levels of knowledge, skills, and attitudes. This more recent model is equally valuable as Brundiers' [2] and is worth considering in and for future guidance. Since this chapter focuses on reporting experiences from the last few years, that specific model had not been available yet to guide junior researchers.

References

1. Abson, D.J., Fischer, J., Leventon, J., Newig, J., Schomerus, T., Vilsmaier, U., Von Wehrden, H., Abernethy, P., Ives, C.D., Jager, N.W., et al.: Leverage points for sustainability transformation. Ambio **46**, 30–39 (2017)
2. Brundiers, K., Barth, M., Cebrián, G., Cohen, M., Diaz, L., Doucette-Remington, S., Dripps, W., Habron, G., Harré, N., Jarchow, M., et al.: Key competencies in sustainability in higher education—toward an agreed-upon reference framework. Sustain. Sci. **16**, 13–29 (2021)
3. Blatti, J.L., Garcia, J., Cave, D., Monge, F., Cuccinello, A., Portillo, J., Juarez, B., Chan, E., Schwebel, F.: Systems thinking in science education and outreach toward a sustainable future. J. Chem. Educat. **96**(12), 2852–2862 (2019)
4. Barrineau, S., Mendy, L., Peters, A.-K.: Emergentist education and the opportunities of radical futurity. Futures **144**, 103062 (2022)
5. Bianchi, G., Pisiotis, U., Cabrera Giraldez, M.: GreenComp: the European sustainability competence framework. Technical report, Joint Research Centre (Seville site), 2022
6. Brown, B.: Dare to Lead: Brave Work. Tough Conversations. Whole Hearts. Random house (2018)
7. Brown, B.: Atlas of the Heart: Mapping Meaningful Connection and the Language of Human Experience. Random House (2021)
8. Coulton, P., Lindley, J.G., Sturdee, M., Stead, M.: Design Fiction as World Building (2017)
9. Fog, K., Budtz, C., Yakaboylu, B.: Storytelling. Springer (2005)
10. Goodwin, J.S., Franklin, S.G.: The beer distribution game: using simulation to teach systems thinking. J. Manag. Dev. **13**(8), 7–15 (1994)
11. Goldman, E.F.: Strategic thinking at the top. MIT Sloan Manag. Rev. **48**(4), 75 (2007)
12. Hilty, L.M., Aebischer, B.: ICT Innovations for Sustainability, vol. 310. Springer (2015)
13. Leventon, J., Abson, D.J., Lang, D.J.: Leverage points for sustainability transformations: nine guiding questions for sustainability science and practice. Sustain. Sci. **16**, 721–726 (2021)
14. Meadows, D.: Leverage points. Places Intervene Syst. **19**, 28 (1999)
15. Penzenstadler, B., Duboc, L., Venters, C.C., Betz, S., Seyff, N., Wnuk, K., Chitchyan, R., Easterbrook, S.M., Becker, C.: Software engineering for sustainability: Find the leverage points! IEEE Software **35**(4), 22–33 (2018)
16. Rockström, J., Sachs, J.D., Öhman, M.C., Schmidt-Traub, G.: Sustainable Development and Planetary Boundaries. JSTOR (2013)
17. Stol, K.-J., Fitzgerald, B.: The ABC of software engineering research. ACM Trans. Software Eng. Methodol. (TOSEM) **27**(3), 1–51 (2018)
18. Sweeney, L.B., Meadows, D.: The Systems Thinking Playbook: Exercises to Stretch and Build Learning and Systems Thinking Capabilities. Chelsea Green Publishing (2010)
19. Sterling, B.: Cover story design fiction. Interactions **16**(3), 20–24 (2009)
20. Ulrich, W.: A mini-primer of boundary critique. Werner Ulrich's Home Page, 1–3 (2005)
21. Wiek, A., Lang, D.J.: Transformational sustainability research methodology. Sustainability Science: An Introduction, 31–41 (2016)
22. Wiek, A., Withycombe, L., Redman, C.L.: Key competencies in sustainability: a reference framework for academic program development. Sustain. Sci. **6**, 203–218 (2011)

Part IV
Teaching Empirical Software Engineering Research Methods

A Course on Experimentation in Software Engineering: Focusing on Doing

Sira Vegas and Natalia Juristo

Abstract Empirical software engineering employs various research methods depending on the research questions, objectives, and the nature of the phenomenon being studied. One of these methods is controlled experiments, which are paramount when the goal is to find causal relationships. This chapter aims to guide educators in designing a course on experimental software engineering. It encompasses the learning goals of the course, the syllabus, the teaching methods, and the assessment activities. Throughout the chapter, we emphasize the lessons we have learned over the years of teaching this course, highlighting both what worked and what did not work.

1 Introduction

Software engineering (SE) technologies (methods, techniques, tools, etc.) must be adequately evaluated. However, there is still little evidence in software development to demonstrate the superiority of one method or technique over another [4, 9, 13, 20]. As a consequence, the software industry still often relies on perceptions, prejudices, and subjective opinions to decide which development technologies to use [3, 11, 18, 19]. That is, professionals are not sure whether a given technology is effective or not, and if it is, they cannot be sure of its level of effectiveness and applicability. In contrast, decision-making in engineering disciplines is characteristically based on objective data [17]. Empirical studies can generate evidence for software development [21]. Applying the empirical paradigm to SE provides a deeper understanding of the software development process and helps uncover the variables that influence it [10].

Empirical SE employs different research methods (or a combination of them) depending on the research questions, the objectives, and the nature of the phenomenon being studied. One of these methods is controlled experiments. In an

S. Vegas (✉) · N. Juristo
Universidad Politécnica de Madrid, Madrid, Spain
e-mail: sira.vegas@upm.es; natalia@fi.upm.es

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2024

D. Mendez et al. (eds.), *Handbook on Teaching Empirical Software Engineering*,
https://doi.org/10.1007/978-3-031-71769-7_11

experiment, key variables of a reality are manipulated in a controlled environment to investigate the impact of such variations [10]. Experiments are the ultimate interventional research method when the goal is to find causal relationships [2].

This chapter aims to assist educators in designing a course on experimental software engineering (ESE). The goal of the course is to equip students with the necessary skills to apply the experimental paradigm within the SE domain. It focuses on the design and analysis of experiments and groups (families) of experiments. The intended audience for this chapter comprises SE educators who are interested in developing a course centered on experimentation. The content of this chapter is drawn from our experience in teaching SE experimentation at the undergraduate, master's, and PhD levels for more than 15 years.

The chapter describes the contents of the course as follows: Section 2 presents specific considerations regarding the organization of the course. Section 3 discusses the recommended prior knowledge for students taking the course. Sections 4 through 7 describe the course contents and explain how the course has evolved over the years through continuous monitoring of what worked and what did not work. Specifically, Sect. 4 enumerates the learning goals of the course, which represent the expected learning outcomes for the students. Section 5 provides a detailed description of the course content (lessons). Section 6 gives an overview of the teaching methods employed in the course. Section 7 describes each assessment activity to be performed. Section 8 lists the teaching materials required for the course (bibliography and other equipment such as software). Finally, Sect. 9 summarizes the lessons learned.

2 Course Description

The body of knowledge related to ESE is vast. Our extensive teaching experience has shown that covering everything in a single course is not feasible. If we attempt this, students will only gain a superficial understanding of each topic and will not acquire the in-depth knowledge required to apply it to real-world situations. Since our ultimate goal is the practical application of the acquired knowledge, we prioritize depth over breadth.

In our experience, a minimum of 18 ECTS[1] are needed to adequately cover all topics related to ESE in sufficient depth. At our university, courses have a maximum duration of 6 ECTS. Therefore, in this chapter, we define three courses, each consisting of 6 ECTS. These courses follow a progression from basic to more complex topics.

1. **Basics of experimental software engineering** aims to establish a foundational understanding of why and how experiments are conducted in SE. Students will

[1] In this chapter, 1 ECTS is equivalent to 8 hours of class time and 27 hours of student work.

learn key concepts and basic experimental designs and data analysis techniques. This course is aimed at undergraduate or master's students.
2. **Advanced experimental software engineering** explores more intricate experimental designs and the advanced statistical methods necessary for their proper analysis. This course is aimed at PhD students.
3. **Groups of software engineering experiments** is dedicated to the topic of conducting multiple experiments and synthesizing their results. This course is aimed at PhD students with a specific profile.

We will use the term *global course* to refer to the set of three courses.

3 Prerequisites

The **basics of experimental software engineering** course has two prerequisites:

- Basic knowledge of SE.
- Basic knowledge of statistics.

A *basic understanding of the principles and practices of SE* is fundamental. Students should be familiar with the basic principles of the software process and life cycles and know the main practices of software development, such as requirements, design, testing, project management, configuration management, or quality assurance. A solid grasp of the fundamentals of SE will provide students with an understanding of the intricacies of the techniques being experimented with. This understanding is necessary to exercise the required level of control to guarantee causality. In addition, it will enable students to properly interpret the experimental results.

Additionally, *basic knowledge of statistics* is essential for students who enroll in this course. As experimentation involves the collection, analysis, and interpretation of data, statistical concepts play a central role in drawing meaningful conclusions. Students should be familiar with fundamental statistical concepts such as distributions, descriptive statistics, estimation, and hypothesis testing. This statistical background will enable students to learn how to conduct inferential statistics and to be able to make informed decisions based on empirical evidence.

By combining SE with statistical knowledge, students will be well equipped to take advantage of experimentation as a powerful tool for continuous improvement and innovation in software development practices.

The **advanced experimental software engineering** course requires the completion of the basics of the experimental software engineering course. Similarly, the **groups of software engineering experiments** course requires the completion of the previous two courses.

4 Objectives and Learning Goals

The general objectives of the global course comprehensively cover the fundamental aspects of ESE.

- GO1: Understand the fundamental principles of ESE.
- GO2: Design and conduct experiments to evaluate SE techniques.
- GO3: Analyze experimental data using appropriate statistical methods.
- GO4: Critically evaluate the validity of experimental results.
- GO5: Interpret experimental results to improve SE practices.
- GO6: Communicate experimental findings effectively.

These general objectives must be operationalized through specific learning goals, which reflect the intended result of a change in student behavior. Since the behaviors targeted by learning can be of different nature (i.e., there are different levels or domains of learning), there are different proposals on how to formulate them. In this work, we have chosen to follow the Gagné approach [6], as it is the most comprehensive.[2] Gagné develops a taxonomy with five categories of human performance established by learning. Table 1 describes these categories, and here we explain how we have incorporated them into our courses. It is important to note that all three courses follow the same structure and progression in the utilization of these categories.

Verbal information is essential in all three courses, as students must learn about the concepts and principles of SE experiments, related to their main elements, the different types of designs that can be used, the statistical methods that should be employed, and the different types of validity threats, among other things. This knowledge forms the core of the courses.

Once students have acquired this foundational knowledge, they can begin developing **intellectual skills**, applying the concepts learned through verbal information to specific tasks related to each activity in the experimental process. The goal is for students to be able to formulate hypotheses, create sound experimental designs, assess the validity threats of an experiment, use appropriate statistical methods to analyze experimental data, interpret experimental results, design a series of related experiments, and synthesize their results to address broader research questions. These are approached from the perspective of single activities in a well-defined context.

Once the intellectual skills are solidified, the courses will move on to developing **cognitive strategies**, where students will learn how to approach the tasks of conducting experiments or groups of experiments (depending on the course) as a whole. This involves not only focusing on individual activities but also dealing with

[2] Other proposals focus solely on one domain of learning: either the cognitive [1] (related to reasoning, thinking, knowledge, understanding, etc.), affective [12] (related to feelings, values, willpower, attitudes, interests, etc.), or the psychomotor [8] (related to voluntary movements of humans, skills, abilities, aptitudes, etc.). However, Gagné's proposal integrates all three domains.

Table 1 Gagné categories of human performance

Category	Description
Verbal information	Acquisition and retention of factual knowledge. Learners are expected to acquire information about facts, concepts, and principles and be able to recall it. Includes what is generally referred to as knowledge.
Intellectual skills	Involves the processing, organization, and handling of knowledge by the learner to solve problems, make decisions, or complete tasks. Refers to having procedural knowledge.
Cognitive strategies	Application of specific mental processes or techniques to guide an individual's learning to improve understanding and retention. Include methods for organizing information, techniques for thinking, ways of analyzing problems, or problem-solving approaches.
Attitudes	Development of learners' feelings, values, and beliefs. Involves fostering positive attitudes, preferences, and predispositions toward certain topics, behaviors, or ways of thinking, so that the learner acquires behavior patterns.
Motor skills	Acquisition and refinement of physical abilities and coordination. Include the development of fine and gross motor skills, such as precise movements or more complex physical activities.

the entire experimentation process, from start to end. Here, students face other types of challenges and will learn methods for identifying and solving issues encountered during the design and analysis of experiments (or groups of experiments).

In parallel with cognitive strategies, the course will work on developing **attitudes**, by fostering a critical evaluation of the design and analysis of experiments and groups of experiments, promoting an attitude of openness to new perspectives when conducting an experiment, cultivating meticulousness and rigor in applying the experimental paradigm, encouraging students to persist through challenges and setbacks, instilling a sense of ethics in conducting experiments, and promoting teamwork and collaboration.

Note that **motor skills** are not covered in our course.

Tables 2, 3, and 4 show the learning goals and their relationship with the levels of Gagné's taxonomy for each of the three courses, respectively. These learning goals have remained largely unchanged over the past 15 years. All were present from the initial conception of the course, except for the *design of groups of experiments* learning goals, which were incorporated some years later, and the *analysis of groups of experiments* learning goals, which were added a few years after the incorporation of the design of groups of experiments learning goals. What has varied is their distribution in ECTS.

Initially, we attempted to cover the learning goals in the *basics of experimental software engineering* course, along with the experimental design learning goals in the *advanced experimental software engineering* course, within a single 6 ECTS course. However, continuous monitoring of student performance consistently showed that they were unable to acquire the learning goals associated with cognitive strategies, and only some were able to acquire those associated with intellectual

Table 2 Learning goals and levels of Gagné taxonomy in the *basics of experimental software engineering* course

Topic	Learning goal	Verbal information	Intellectual skills	Cognitive strategies	Attitudes
Foundations	LG1. Define what experimentalism is	X			
	LG2. Define what a cause-effect relationship is	X			
	LG3. Be aware of the scientific immaturity of SE	X			
	LG4. Define the SE laboratory	X			
	LG5. Explain what a SE experiment is	X			
	LG6. Describe the basic elements of an experiment	X			
	LG7. Perform the operationalization of an experiment		X		
Experimental design	LG8. Explain the fundamental strategies for designing experiments	X			
	LG9. Apply the fundamental strategies to design an experiment		X		
	LG10. Describe the within- and between-subjects designs	X			
	LG11. Explain what power analysis is	X			
	LG12. Choose the best design for an experiment		X		
	LG13. Be aware of what validity of an experiment is	X			
	LG14. Explain each category of validity threat	X			
	LG15. Assess the validity of an experiment		X		

Statistical methods	LG28. Explain the existing measurement scales	X				
	LG29. Describe the null hypothesis statistical testing paradigm	X				
	LG30. Define the concept of statistical tests assumptions	X				
	LG31. Describe parametric tests for two treatments	X				
	LG32. Apply parametric tests for two treatments		X			
	LG33. Describe parametric tests for more than two treatments	X				
	LG34. Apply parametric tests for more than two treatments		X			
	LG35. Perform non-parametric tests		X			
	LG36. Perform data transformation		X			
Transversals	LG52. Interpret the results of the statistical analysis		X			
	LG53. Elaborate the report of an experiment		X			
	LG54. Create the design of an experiment			X		X
	LG55. Elaborate the analysis of an experiment			X		X

Table 3 Learning goals and levels of Gagné taxonomy in the *advanced experimental software engineering* course

Topic	Learning goal	Verbal information	Intellectual skills	Cognitive strategies	Attitudes
Experimental design	LG16. Describe factorial designs	X			
	LG17. Develop a factorial experiment design		X		
	LG18. Explain what a blocked design is	X			
	LG19. Develop a blocked design		X		
	LG20. Describe nested designs	X			
	LG21. Apply a nested design		X		
	LG22. Describe crossover designs	X			
	LG23. Apply a crossover design		X		
Statistical methods	LG37. Define the statistical concept of interaction	X			
	LG38. Describe the different types of interactions	X			
	LG39. Apply the full factorial ANOVA		X		
	LG40. Apply the repeated measures full factorial ANOVA		X		
	LG41. Define the statistical implications of fractional factorial analyses	X			
	LG42. Apply the fractional factorial ANOVA		X		
	LG43. Apply the repeated measures fractional factorial ANOVA		X		
	LG44. Perform the analysis of a blocked design		X		
	LG45. Perform the analysis of a nested design		X		
	LG46. Perform the analysis of a crossover design		X		
Transversals	LG52. Interpret the results of the statistical analysis		X		
	LG53. Elaborate the report of an experiment		X		
	LG54. Create the design of an experiment			X	X
	LG55. Elaborate the analysis of an experiment			X	X

Table 4 Learning goals and levels of Gagné taxonomy in the *groups of software engineering experiments* course

Topic	Learning goal	Verbal information	Intellectual skills	Cognitive strategies	Attitudes
Experimental design	LG24. Explain what a replication is	X			
	LG25. List the types of replications that exist	X			
	LG26. Describe the concept of a family of experiments	X			
	LG27. Design a family of experiments		X		
Statistical methods	LG47. Explain the concept of meta-analysis	X			
	LG48. List the different types of meta-analysis	X			
	LG49. Understand when to apply each meta-analysis approach	X			
	LG50. Perform AD meta-analysis		X		
	LG51. Perform IPD meta-analysis		X		
Transversals	LG56. Interpret the results of the statistical analysis		X		
	LG57. Elaborate the report of a family of experiments		X		
	LG58. Create the design of a family of experiments			X	X
	LG59. Elaborate the analysis of a family of experiments			X	X

skills. We attributed this to the teaching method used (to be discussed in Sect. 6), but it turned out to be only part of the problem. Changing the teaching method improved the acquisition of intellectual skills, but students still struggled to acquire the cognitive strategies. We then tried reducing the number of learning goals to be covered by breaking down the course, which proved to be more effective.

5 Syllabus

Building on the description from the previous section, the content of the courses has been organized into different topics, with their content grouped according to their respective themes and categorized into 16 lessons. In this way, the framework for the development of the courses is as follows (Appendix: Detailed Syllabus shows the detailed syllabus):

Part I. Foundations

1. *Introduction to Experimental Software Engineering.* It starts with a discussion on the basics of experimentalism, providing students with an understanding of the principles underlying experimental disciplines. Then it explores cause-effect relationships as they pertain to experiments. The concept of scientific immaturity within the realm of SE is also addressed, shedding light on the evolving nature of methodologies and practices in the field.
2. *What Is an Experiment*. This lesson provides a comprehensive exploration of key concepts related to experimental practices. The lesson starts with the fundamental concept of an experiment, elucidating its essential components. Then, the lesson presents the concept of a SE experiment, providing students with examples to reinforce their understanding. The lesson then delves into examining the concept of a laboratory, offering insights into its significance as a controlled environment for conducting experiments. Finally, it discusses the notion of a laboratory specifically tailored for SE experiments, emphasizing the importance of creating controlled settings to observe, analyze, and draw conclusions in this dynamic domain.
3. *Basic Elements of an Experiment*. This lesson systematically introduces the basic components of an experiment. It begins by elucidating the concept of response variables and their operationalization. The lesson then explores the significance of factors and treatments along with their operationalization. Additionally, it draws a distinction between experiments and quasi-experiments, highlighting the characteristics that differentiate these research designs.

Part II. Experimental Design

4. *Design Strategies*. This lesson introduces the three fundamental strategies for designing controlled experiments: keeping variables constant, randomization, and blocking. It also explores the situations in which each of these strategies is valid and the types of variables that they allow control over. In addition, it discusses sampling and participant selection.
5. *Types of Designs*. This lesson provides a comprehensive exploration of different experimental design approaches in the context of one-factor experiments. It elucidates the concept of between- and within-subjects designs and introduces the concept of power analysis. The lesson equips students with a nuanced understanding of how to strategically structure experiments, considering how the variability and constraints inherent in different contexts affect their design. Lastly, it covers how the experimental design must be reported.
6. *Validity Threats*. This lesson aims to identify and address those factors that may compromise the reliability and validity of experimental results. The categorization of validity threats is presented, and each one (conclusion, internal, construct and external) is discussed. It introduces the categorization of validity threats, and each category (conclusion, internal, construct, and external) is thoroughly discussed.
7. *Factorial Designs*. This lesson explores the intricacies of the design of factorial designs. It starts by introducing the concepts of main effects and interaction in experiments with more than one factor. Then, the lesson covers full factorial designs and fractional factorial designs, which offer a more resource-efficient alternative when dealing with a large number of factors. Finally, it presents how these types of experimental designs must be reported.
8. *Advanced Designs*. This lesson delves into more sophisticated experimental design strategies. It starts by exploring the concept of blocked designs, where experimental units are grouped or blocked on the basis of specific characteristics before the application of treatments. Then, the lesson introduces nested designs, which involve the incorporation of hierarchical structures within experimental units. Later, crossover designs are introduced, in which participants experience different treatments at different times. Lastly, it covers how advanced designs must be reported.
9. *Replications*. This lesson begins by covering the critical concept of replicating experiments in the SE domain, emphasizing the importance of running more than one experiment to obtain reliable results. The lesson then explores the different types of replications that can take place in SE. Finally, it covers how replications must be reported.

Part III. Statistical Methods

10. *Introduction to Data Analysis*. This lesson provides a comprehensive exploration of key concepts essential to understanding and conducting data analysis in the context of SE. It begins by elucidating the concept of measurement scales, providing insights into the different levels of measurement and their implications for data analysis. The lesson then introduces statistical models, emphasizing their role in representing relationships within datasets. It goes beyond raw data by addressing strategies for comprehensive analysis, exploring topics such as hypothesis testing for statistical significance, effect size, and statistical power. In addition, the lesson offers an overview of the statistical tool to be used during the course in the analysis process.
11. *Which Is the Better of Two Treatments?* This lesson is dedicated to exploring statistical methods to compare two treatments. It introduces the student's t-test to assess whether there is a significant difference between the means of two independent groups. The lesson further delves into the paired-samples t-test, a variant designed for situations where observations are paired. Students are offered practical insight into the application of these techniques. They are also taught how to run the analyses with a statistical tool, how to compute and interpret power and effect size, and how experimental results should be reported.
12. *Which of K Treatments Is the Best?* This lesson is dedicated to exploring statistical techniques to compare the means of multiple treatment groups. It introduces the one-way analysis of variance (ANOVA). The lesson also delves into the repeated measures one-way ANOVA, a variant suitable for situations where measurements are taken on the same subjects over multiple time points or conditions. Students are offered practical insight into the application of these techniques. Finally, it is explained how to run the analyses with a statistical tool, how to compute and interpret power and effect size, and how experimental results should be reported.
13. *When Statistical Tests Assumptions Are Not Met*. This lesson addresses the scenario where the assumptions for traditional parametric statistical tests are not met. It explores alternative strategies to handle such situations, emphasizing two key approaches: data transformation and the use of non-parametric tests. Students are offered practical insight into the application of these techniques. Lastly, they are taught how to run the analyses with a statistical tool, how to compute and interpret power and effect size, and how experimental results should be reported.
14. *Analyzing Experiments with Several Factors*. This lesson provides a comprehensive exploration of advanced statistical techniques tailored for experiments that involve multiple factors. It introduces the full and fractional factorial ANOVA, which are powerful analytical tools for investigating the main effects and interactions among factors in experimental designs. The lesson also covers the repeated measures full and fractional factorial ANOVAs, extending the analysis to scenarios where measurements are collected on the same subjects

across different conditions or time points. Students are offered practical insight into the application of these techniques. Finally, the topics of how to run the analyses with a statistical tool, how to compute and interpret power and effect size, and how experimental results should be reported are covered.

15. *Analyzing Advanced Experiments*. This lesson offers a comprehensive exploration of sophisticated statistical techniques tailored for the analysis of complex experimental designs. It covers the analysis of blocked designs, which aids in controlling variability and enhancing the precision of experimental results. The lesson further delves into the analysis of nested designs, which considers hierarchical structures within experimental units Then, the lesson explores the analysis of crossover designs, providing insights into within-subject variations. Students are offered practical insight into the application of these techniques. Finally, students are taught how to run analyses with a statistical tool, how to compute and interpret power and effect size, and how experimental results should be reported.
16. *Analyzing Groups of Experiments*. This lesson equips students with advanced techniques to comprehensively analyze groups of SE experiments and synthesize their findings into a cohesive result. It begins by emphasizing the importance of reliably obtaining joint results. The lesson then explores two distinct approaches to meta-analysis: aggregate data (which involves synthesizing summarized results from multiple studies and provides a broader perspective on the overall findings) and individual participant data (which delves into the raw, individual-level data, allowing for more detailed and nuanced insights). Lastly, students are taught how to run the analyses with a statistical tool, how to compute and interpret power and effect size, and how joint results should be reported.

The syllabus of the courses has evolved in parallel with the learning goals. Incorporating new learning goals has led to the addition of new content, and organizing learning goals into separate courses has resulted in the reorganization of content within those courses.

6 Teaching Method

The teaching method is the set of didactic procedures aimed at guiding learning. It is applied through teaching techniques. Table 5 describes the teaching techniques used in our courses. In the following, we provide a description of how these techniques have been incorporated into them.

Table 6 details the teaching techniques used in each lesson.

The first (introductory) lesson is conducted in a traditional lecture style, while the second lesson adopts a flipped classroom approach to engage and motivate the students. Subsequent design lessons follow the same structure. Each lesson covers various topics, each topic beginning with a lecture on the main theoretical

concepts. As these lectures may lack interaction, the professor supplements them with questions to ensure that students grasp the verbal information effectively. Following this, a second lecture delves deeper into the topic, allowing students to apply the learned concepts to their projects using the project-based learning technique and working in small groups.

The introduction to data analysis is once again delivered in a traditional lecture style. Subsequent data analysis lessons maintain the same structure. Each statistical test is introduced with an explanation of its purpose, the assumptions it makes regarding the data, and how to interpret its results, illustrated with examples tailored to the specific test. Following this, the students are engaged in several small problems to apply the learned concepts in pairs. In addition, outside of the

Table 5 Teaching techniques

Technique	Description
Lecture	The professor presents information to the students in a structured manner, often through spoken words. It is a traditional method that is commonly used in higher education.
Questioning	Formulation of knowledge or memorization questions, with an emphasis on comprehension questions. It has a cognitive approach, used to assess the development of the student's learning process, educational objectives, and the teacher's own performance, and an affective approach, used to facilitate learning and enhance interaction between the teacher and the student.
Flipped classroom	Inverts traditional teaching methods by delivering instructional content online for students to review outside of class, while class time is used for interactive discussions and activities.
Problem-solving	Small examples of increasing difficulty are initially solved by the teacher and later by the students, dealing with different levels of complexity. These comprehensive problems require significant challenges to be overcome using experiences and knowledge acquired during the course. This approach helps students develop an overview, connect seemingly distant issues, and introduces them to analysis and synthesis techniques, as well as simplification, generalization, abstraction, and concretization.
Example-based learning	Presentation of concrete examples to illustrate concepts, principles, or problem-solving techniques. Examples can serve as models for understanding and application. The teacher may provide worked-out examples and then guide students in solving similar problems.
Class projects	Students collaborate on a specific assignment or task, applying knowledge and skills acquired during the course to complete a project. The scope and format of the project vary, as is designed to reinforce learning, encourage teamwork, and allow students to demonstrate their understanding. The teacher may guide the structure and objectives, with the assessment often evaluating both the process and the final product or presentation.
Project-based learning	Students work on an extended project to explore and apply concepts in a more hands-on and practical way. It emphasizes student-driven inquiry, real-world application, and the development of essential skills.

Table 6 Relationship between lessons and teaching techniques

Topic	Lesson	Teaching techniques
Foundations	1. Introduction to experimental SE	Lecture
	2. What is an experiment	Flipped classroom
	3. Basic elements of an experiment	Lecture
		Questioning
		Project-based learning
Experimental design	4. Design strategies	Lecture
		Questioning
		Project-based learning
	5. Types of designs	Lecture
		Questioning
		Project-based learning
	6. Validity threats	Lecture
		Questioning
		Project-based learning
	7. Factorial designs	Lecture
		Questioning
		Project-based learning
	8. Advanced designs	Lecture
		Questioning
		Project-based learning
	9. Replications	Lecture
		Questioning
		Project-based learning
Statistical methods	10. Introduction to data analysis	Lecture
	11. Which is better of two treatments?	Example-based learning
		Problem solving
		Class projects
	12. Which of k treatments is the best?	Example-based learning
		Problem solving
		Class projects
	13. When tests assumptions are not met	Example-based learning
		Problem solving
		Class projects
	14. Analyzing experiments with several factors	Example-based learning
		Problem solving
		Class projects
	15. Analyzing advanced experiments	Example-based learning
		Problem solving
		Class projects
	16. Analyzing groups of experiments	Example-based learning
		Problem solving
		Class projects

classroom, students work in small groups on a class project, where they analyze and interpret the results of multiple experiments inspired by real-world experiments but using artificially created data.

It is important to note that while the course content has remained relatively constant over the years (as detailed in Sect. 5), our approach to teaching has undergone significant changes. We can identify four distinct periods or stages:

1. **First period.** We relied on traditional lecture-style teaching for all lessons, supplemented by a class project for experimental design and small exercises (problem-solving technique) for analysis. In addition, no statistical tool was utilized for data analysis.

   However, student performance in the course was consistently poor during this period. They struggled to grasp the learning goals associated with cognitive strategies, and only a fraction acquired the learning goals associated with intellectual skills. The students complained that the statistical methods part was too difficult.

   We decided to act on the teaching method to address the identified problems. We implemented changes primarily focused on the statistical methods part.
2. **Second period.** The traditional lecture format remained the primary method for transmitting information to students. The separate class project and small problem-solving tasks used in the design and analysis parts were combined into a single project. Additionally, a statistical tool for data analysis, IBM SPSS, was introduced. This choice was made because our university provides free access to students, and we used this to remove the requirement for prior programming knowledge (unlike R or Python).[3]

   Using a statistical tool for analysis certainly proved to be a wise choice. However, merging the design class project and small data analysis problems into a single project turned out to be less advantageous. The project involved designing, conducting, and analyzing the results of an experiment chosen by the students (with groups consisting of 4–5 individuals). Although this approach suited the design aspect well, students encountered difficulties in conducting the experiments consistently each year. Many reported severe issues that resulted in poor quality data or insufficient data points. Consequently, this hindered the data analysis phase.

   At this point, we decided to address the identified issues by making changes to both the teaching method and the organization of the course. Some of these changes included reverting to previous teaching techniques.
3. **Third period.** The course is structured as explained in Sect. 4. Design and data analysis projects are now separated. Project-based learning is adopted for design, given the success of the project approach. However, for analysis, we have introduced class projects that involve the analysis of fabricated data (generated by the professor) simulating real-world experiments. This approach is expected

[3] Due to the diverse backgrounds of our students, proficiency in specific programming languages cannot be assumed.

to be an improvement over the small problems used in the first period, which were limited by the lack of statistical tools.

These changes proved effective; however, we identified two areas for improvement. Firstly, students were not fully meeting the learning objectives related to verbal information. Secondly, due to the tool's use, students struggled to distinguish between the generic statistical concepts and the specific functionalities of the tool.

These observations prompted us to make some minor adjustments, leading us to the fourth (and current) period, as elaborated above.

7 Assessment Activities

Any educational proposal must be accompanied by an evaluation procedure to determine to what extent the proposal meets its intended purposes. For this reason, it is necessary to detail the evaluation instruments that will be used. The instruments proposed in the literature include written tests, oral tests, practical tests, observation, and questionnaires. Each is more suitable for assessing a different learning domain, as shown in Table 7.

Table 8 describes the assessment activities used in our courses. Below, we provide an overview of how these proposed assessment activities are utilized. Table 9 shows the relationship between lessons and assessment activities.

- **Lessons 1, 2, and 10**. A closed written test (in the form of a quiz) is administered after each lesson.
- **Lessons 3–9**. After each lesson, there is a closed written test (quiz). Additionally, a practical test is conducted where students work in groups,[4] designing an experiment (or family of experiments) using the project-based learning approach. During these lessons, time is allocated in each lecture for an oral test as a follow-up to the project.

Table 7 Instruments available to assess learning domains

Domain	Instruments
Cognitive	Written tests
	Oral tests
	Practical tests
Affective	Observation
	Practical tests
	Questionnaires
Psychomotor	Practical tests
	Observation

[4] Depending on the size of the class, they could work individually.

- **Lessons 11–16**. After each lesson, there is a closed written test (quiz). In addition, a practical test is conducted where students work in groups[4] to analyze data from experiments (or family of experiments) following the class project teaching technique. Throughout these lessons, time is allocated in each lecture to address questions and issues related to the project.

Table 8 Assessment activities

Activity	Description
Written tests	There are two types: objective (or closed) and open. Objective tests consist of questions that have pre-established, unique answers, characterized by their objectivity and convergence. These can include simple answer questions, multiple-choice, matching (or association), or sorting. Open tests, on the other hand, require students to provide elaborated responses to questions.
Oral tests	Oral examinations have historically been the primary method for assessing learning or performance, remaining predominant in schools until the last century. Traditionally used indiscriminately for all types of assessment, their appropriate application is limited to certain cognitive objectives, such as oral expression, fluency, and communication. In the context of university teaching, it is essential to include activities requiring students to speak publicly.
Practical tests	Achieving specific educational objectives requires not only theoretical knowledge but also the ability to apply this knowledge practically. The goal is for learners to demonstrate their capability to use their knowledge in real-world situations. These action-oriented objectives can be assessed through observation of practical activities, evaluating both the processes and the final products.

Table 9 Relationship between lessons and assessment activities

		Test		
Topic	Lesson	Written	Oral	Practical
Foundations	1. Introduction to experimental SE	X		
	2. What is an experiment	X		
	3. Basic elements of an experiment	X	X	X
Experimental design	4. Design strategies	X	X	X
	5. Types of designs	X	X	X
	6. Validity threats	X	X	X
	7. Factorial designs	X	X	X
	8. Advanced designs	X	X	X
	9. Replications	X	X	X
Statistical methods	10. Introduction to data analysis	X		
	11. Which is better of two treatments?	X		X
	12. Which of k treatments is the best?	X		X
	13. When tests assumptions are not met	X		X
	14. Analyzing experiments with several factors	X		X
	15. Analyzing advanced experiments	X		X
	16. Analyzing groups of experiments	X		X

The assessment activities used in the courses have evolved alongside the teaching method. During the **first period**, the design part was evaluated with a practical test, assessing the project developed by the students, and an oral test where the students presented their projects. The analysis part was evaluated through a written test. In the **second period**, the assessment activities were adjusted to fit the teaching method. Both the design and analysis parts were evaluated with a practical test, assessing the project developed by the students, and an oral test where the students presented their projects. In the **third period**, the assessment activities were slightly adjusted again to fit the new teaching method. While the design part was evaluated in the same way as in the second period, the analysis part is now evaluated with a practical test only, assessing the project developed by the students. In the **fourth period**, the assessment activities were changed once more to address the issue of students not fully meeting the learning objectives related to verbal information, resulting in the activities described above.

8 Teaching Resources

Teaching resources are material elements whose function is to facilitate the communication established between educators and learners. They can also be used to stimulate learning. The teaching resources used in this course are:

- *Slides*. They are used to complement the following teaching techniques: lecture, example-based learning, and problem-solving.
- *Books*. Three books serve as the main bibliography for the *basics of experimental software engineering* and *advanced experimental software engineering* courses: Juristo and Moreno [10] for the foundations and design parts, Shadish et al. [16] for validity threats, and Field [5] for the analysis part.
- *Papers*. Three papers serve as the main bibliography for the *groups of software engineering experiments*: Gómez et al. [7], Santos et al. [14], and Santos et al. [15].

  Additional research papers can be used during the application of the example-based learning technique in the data analysis part, to show real application examples.
- *Moodle*. The Moodle site is utilized in multiple ways. Professors use it to share course materials (slides and papers), while students use it to complete the quizzes and submit their projects.
- *Statistical analysis tool*. For the reasons mentioned above, we have chosen IBM SPSS as our statistical analysis tool for the data analysis part.

9 Lessons Learned

Figure 1 shows the evolution of the course since its inception. Next, we summarize the main lessons we have learned over the years.

With respect to the **content** of the course, we have learned that depth is better than breadth. When students lack previous knowledge about experimentation, it is more effective for them to learn a few well-selected core learning goals rather than a large amount of information. For example, in the case of data analysis, it is more important for students to understand the importance of statistical assumptions of tests rather than learning a wide variety of tests. In the case of design, it is better to teach core concepts of experimental design (i.e., manipulation, blocking, randomization, etc.) rather than complex designs (e.g., nested designs).

Regarding the **teaching method**, we have learned that the flipped classroom is an interesting approach to the introductory lesson, as it enhances student motivation. We have also learned that the classical lecture format supplemented with examples (example-based learning) and questioning techniques is valuable for transmitting core concepts to students. However, for acquiring more complex or elaborated concepts, project-based techniques have proven superior. Specifically, project-based learning is well suited for the design part, allowing for in-depth exploration of different design approaches within a single experiment. Conversely, the data analysis part benefits more from students encountering various scenarios, which is better facilitated through a problem-solving strategy combined with class projects with synthetic data. In the case of data analysis, the course requires the use of a statistical analysis tool; if students do not have programming language skills (e.g., R, Python, etc.), we strongly recommend using graphical statistical tools instead (e.g., IBM SPSS statistics, Statgraphics, etc.). Lastly, while the ideal teaching approach would involve students running an experiment from start to end, we acknowledge that we have not been able to successfully implement this task.

Finally, in relation to **assessment activities**, we have learned that practical tests are essential, as they are the only means to verify that students have acquired the learning goals at the cognitive level. Oral tests are also of utmost importance. Their usefulness lies in two aspects: first, they are valuable for evaluating learning goals at the attitudinal level; second, they are helpful when practical evaluation is performed in groups rather than individually, as they can help elucidate each student's participation and/or contribution. Additionally, we strongly encourage the use of small written tests (in the form of quizzes) to check that students have learned the basic terminology and core concepts of the course.

Acknowledgments This work was supported by grant PID2022-137846NB-I00 funded by MCIN/AEI/10.13039/501100011033, and by "ERDF A way of making Europe."

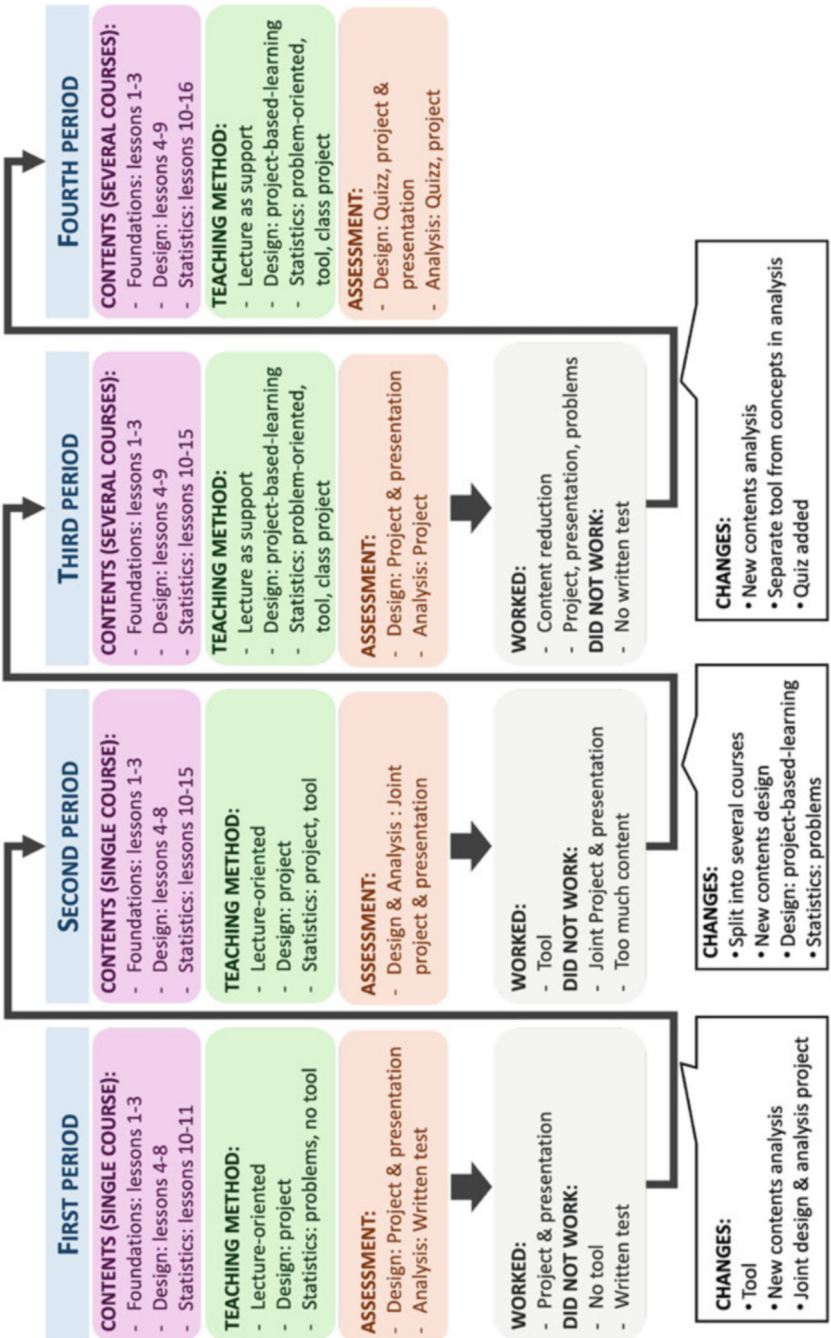


Fig. 1 Evolution of the experimental software engineering course

Appendix: Detailed Syllabus

Part I. Foundations

1. Introduction to Experimental Software Engineering
   - 1.1. Basics of Experimentalism
   - 1.2. The Scientific Method
   - 1.3. Scientific Rules: Cause-Effect Relationships
   - 1.4. Scientific Immaturity of Software Engineering
2. What Is an Experiment
   - 2.1. The Concept of Experiment
   - 2.2. An Experiment for Software Engineering
   - 2.3. The Concept of Laboratory
   - 2.4. A Laboratory for Software Engineering
3. Basic Elements of an Experiment
   - 3.1. Response Variables
   - 3.2. Factors and Treatments
   - 3.3. Experiments vs. Quasi-experiments

Part II. Experimental Design

4. Design Strategies
   - 4.1. Held-Constant
   - 4.2. Randomization
   - 4.3. Blocking
   - 4.4. Sampling and Participants Selection
5. Types of Designs
   - 5.1. Between/Within Subjects
   - 5.2. Power Analysis
6. Validity Threats
   - 6.1. Types of Validity Threats
   - 6.2. Conclusion and Internal Validity Threats
   - 6.3. Construct and External Validity Threats
7. Factorial Designs
   - 7.1. Full Factorial
   - 7.2. Fractional Factorial
8. Advanced Designs
   - 8.1. Blocked Designs
   - 8.2. Nested Designs

8.3. Crossover Designs

9. Replications

9.1. Why Replication Is Important
9.2. Types of Replications

Part III. Statistical Methods

10. Introduction to Data Analysis

10.1. Measurement Scales
10.2. Statistical Models
10.3. Going Beyond the Data
10.4. Hypothesis Testing
10.5. Statistical Significance, Effect Size, and Power
10.6. Overview of the Statistical Tool Used

11. Which Is the Better of Two Treatments?

11.1. Student's t-Test
11.2. Paired Samples t-Test

12. Which of K Treatments Is the Best?

12.1. One-Way ANOVA
12.2. Repeated Measures One-Way ANOVA

13. When Statistical Tests Assumptions Are Not Met

13.1. Data Transformation
13.2. Non-parametric Tests

14. Analyzing Experiments with Several Factors

14.1. (Fractional) Factorial ANOVA
14.2. Repeated Measures (Fractional) Factorial ANOVA

15. Analyzing Advanced Experiments

15.1. Analysis of Blocked Designs
15.2. Analysis of Nested Designs
15.3. Analysis of Crossover Designs

16. Analyzing Groups of Experiments

16.1. Aggregate Data (AD) Meta-analysis
16.2. Individual Participant Data (IPD) Meta-analysis

References

1. Bloom, B.S.: Taxonomy of Educational Objectives. The Classification of Educational Goals. Handbook I: Cognitive Domain. Longmans (1956)
2. Chambliss, D., Schutt, R.: Making Sense of the Social World, 6th edn. Sage Publications (2018)
3. Devanbu, P., Zimmermann, T., Bird, C.: Belief & evidence in empirical software engineering. In: Proceedings of the 38th International Conference on Software Engineering, ICSE '16, pp. 108–119. Association for Computing Machinery, New York (2016). https://doi.org/10.1145/2884781.2884812
4. Dybå, T., Bergersen, G., Sjøberg, D.: Evidence-Based Software Engineering, pp. 149–153. Morgan Kaufmann, Los Altos (2016)
5. Field, A., Miles, J., Field, Z.: Discovering Statistics Using R. Sage Publications (2012)
6. Gagne, R.M.: The Conditions of Learning and Theory of Instruction, 4th edn. Holt, Rinehart & Winston (1985)
7. Gómez, O.S., Juristo, N., Vegas, S.: Understanding replication of experiments in software engineering: A classification. Inform. Softw. Technol **56**(8), 1033–1048 (2014). https://doi.org/10.1016/j.infsof.2014.04.004
8. Harrow, A.J.: A Taxonomy of the Psychomotor Domain: A Guide for Developing Behavioral Objectives. David McKay Company (1972)
9. Juristo, N., Moreno, A.: Reliable knowledge for software development. IEEE Softw. **19**(5), 98–99 (2002)
10. Juristo, N., Moreno, A.M.: Basics of software engineering experimentation. Springer, Berlin (2011)
11. Kotti, Z., Gousios, G., Spinellis, D.: Impact of software engineering research in practice: A patent and author survey analysis. IEEE Trans. Softw. Eng. **49**(04), 2020–2038 (2023). https://doi.org/10.1109/TSE.2022.3208210
12. Krathwohl, D.R., Bloom, B.S., Masia, B.B.: Taxonomy of Educational Objectives. The Classification of Educational Goals. Handbook 2: Affective Domain. Longmans (1964)
13. Pfleeger, S.: Albert einstein and empirical software engineering. Computer **32**(10), 32–38 (1999). https://doi.org/10.1109/2.796106
14. Santos, A., Vegas, S., Oivo, M., Juristo, N.: Comparing the results of replications in software engineering. Empir. Softw. Eng. **26**(1), 13 (2021). https://doi.org/10.1007/S10664-020-09907-7
15. Santos, A., Vegas, S., Oivo, M., Juristo, N.: A procedure and guidelines for analyzing groups of software engineering replications. IEEE Trans. Softw. Eng. **47**(9), 1742–1763 (2021). https://doi.org/10.1109/TSE.2019.2935720
16. Shadish, W.R., Cook, T.D., Campbell, D.T.: Experimental and Quasi-Experimental Designs for Generalized Causal Inference. Wadsworth, Cengage Learning (2002)
17. Shaw, M.: Prospects for an engineering discipline of software. IEEE Softw. **7**(6), 15–24 (1990). https://doi.org/10.1109/52.60586
18. Shaw, M.: Continuing prospects for an engineering discipline of software. IEEE Softw. **26**(6), 64–67 (2009). https://doi.org/10.1109/MS.2009.172
19. Shaw, M.: Progress toward an engineering discipline of software. In: 2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C), pp. 3–4 (2016). http://2016.icse.cs.txstate.edu/static//downloads/docs-and-slides/mary-shaw-keynote-slides.pdf
20. Wohlin, C., Rainer, A.: Challenges and recommendations to publishing and using credible evidence in software engineering. Inform. Softw. Technol. **134**, 106555 (2021). https://doi.org/10.1016/j.infsof.2021.106555
21. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in software engineering. Springer, Berlin (2012)

Teaching Mining Software Repositories

Zadia Codabux, Fatemeh Fard, Roberto Verdecchia, Fabio Palomba, Dario Di Nucci, and Gilberto Recupito

Abstract Mining software repositories (MSR) has become a popular research area recently. MSR analyzes different sources of data, such as version control systems, code repositories, defect tracking systems, archived communication, deployment logs, and so on, to uncover interesting and actionable insights from the data for improved software development, maintenance, and evolution. This chapter provides an overview of MSR and how to conduct an MSR study, including setting up a study, formulating research goals and questions, identifying repositories, extracting and cleaning the data, performing data analysis and synthesis, and discussing MSR study limitations. Furthermore, the chapter discusses MSR as part of a mixed method study and how to mine data ethically and gives an overview of recent trends in MSR as well as reflects on the future. As a teaching aid, the chapter provides tips for educators, exercises for students at all levels, and a list of repositories that can be used as a starting point for an MSR study.

1 Introduction

Embedded within the intricate software development landscape lies mining software repositories (MSR), an academic discipline dedicated to delving into the expansive

Z. Codabux (✉)
University of Saskatchewan, Saskatoon, SK, Canada
e-mail: zadiacodabux@ieee.org

F. Fard
University of British Columbia, Kelowna, BC, Canada
e-mail: fatemeh.fard@ubc.ca

R. Verdecchia
University of Florence, Firenze, Italy
e-mail: roberto.verdecchia@unifi.it

F. Palomba · D. Di Nucci · G. Recupito
University of Salerno, Salerno, Italy
e-mail: fpalomba@unisa.it; ddinucci@unisa.it; grecupito@unisa.it

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2024

D. Mendez et al. (eds.), *Handbook on Teaching Empirical Software Engineering*,
https://doi.org/10.1007/978-3-031-71769-7_12

dataset within software repositories. Software repositories are among the most common sources of data that enable the analysis of software properties. Additionally, the promising success that this type of study had in the research community allowed to extend the mining data process beyond the software repositories, encompassing various essential aspects such as software changes, collaborative efforts, socio-technical aspects [25, 40], and the process dynamics [45]. Combining these aspects serves to obtain the source of key information detailing the transformation of a software application from its preliminary stages to well-defined and sophisticated software.

During the iterative enhancement of software development, developers wield powerful tools that meticulously track software information. These tools, including robust version control systems like Git and Subversion (SVN) and efficient issue-tracking tools like JIRA and Bugzilla, facilitate collaboration and provide an exhaustive lineage of code changes. This comprehensive history, resulting from the union of these tools, allows developers to gain profound insights into all key aspects of the software evolution project, including software modifications, issue resolutions, bug fixes, and strategic refactoring initiatives, which are contained in the software repository [32]. The definition of this dynamic and extensive source of information lays the basis for introducing MSR. MSR is the field that exploits the information contained in software repositories to conduct thorough investigation and analysis, unveiling aspects of the software development process [28].

MSR studies in the software engineering research field demonstrated the underlying capabilities of repository analysis. Several studies investigate the possibility of building prediction models for software quality issues. Prediction models could be built from issue tracking labels to detect bugs [39, 48]. Moreover, the extreme amount of data possible to extract from source code repositories allows us to build complex and useful models with the use of large language models, enabling us to build useful tools that perform programming tasks related to code generation, summarization, and analysis, also producing code that is not complex and highly maintainable (e.g., Copilot[1]) [44]. This chapter is intended to **target educators at MSc and PhD levels**, providing them with a comprehensive understanding of MSR and its applications in software engineering research so that they may be equipped to teach this method. More specifically, **learning objectives** include:

1. Understand **how to set up a mining software repository study**, uncovering the methodological design steps required to (i) define the objectives of an MSR study, (ii) identify suitable data sources and data cleaning methods to address the objectives, (iii) select the most appropriate code analysis instruments to extract relevant pieces of information useful to the analysis, and (iv) analyze and synthesize the data gathered throughout the mining process

[1] GitHub Copilot: https://github.com/features/copilot/

2. Realize **what the typical threats to the validity are involving mining software repository studies**, overviewing how multiple issues concerned with the mining and analysis of data sources might bias the interpretation of the results
3. Identify **how to combine mining software repository studies with additional research methods**, discussing the flexibility of MSR research in addressing complex research objectives
4. Indicate **what the ethical considerations are concerned with mining software repository studies**, uncovering the possible limitations of this research method, other than the responsibilities that the future generation of researchers must necessarily take into account when designing similar studies

To ease the reader's understanding of the concepts in the chapter, we will focus on specific cases regarding extracting data from the source code of software repositories. Specifically, we will often refer to an exemplar case, which effectively used the methodologies described in the chapter. Among the vast array of valuable MSR examples in literature, we decided to focus on the work by Kamei et al. [34], published in *IEEE Transactions on Software Engineering* in 2012.

The study explored a well-known and well-established research theme in MSR research, namely, *defect prediction*. This is a technique used to identify defect-prone files or packages throughout software development. More particularly, the study introduced the term *"Just-In-Time Quality Assurance"* and aimed at identifying defect-prone software changes while the developers commit their changes onto a shared repository. From a methodological standpoint, the study heavily relied on MSR instruments, as it was required to feed machine-learning models with features mined from software repositories. These features encompassed multiple properties that might be extracted from version control systems, such as product, process, and developer-oriented metrics, hence representing an ideal example to make the concepts in this chapter more practical. In addition, the study carefully designed the experimental procedures, touching on various data selection, cleaning, and analysis processes that are elaborated upon in this chapter.

Exemplary **teaching strategies** that might be relevant for teaching this chapter include, but are not limited to, the following:

- *Interactive discussions*. Educators may engage students in active discussions about the importance of MSR in software engineering research, its potential applications, and its limitations. Through these discussions, we envision students to acquire awareness of when and how to use MSR instruments to empower their analyses.
- *Case studies*. Educators may use our chapter to present real-world examples of MSR research, such as the study by Kamei et al. [34], to illustrate key concepts and methodologies, making students aware of how the design choices taken in exemplary cases impacted the type of analysis and results of MSR studies.
- *Hands-on exercises*. Educators may provide students with hands-on experience in conducting MSR studies, including data collection, cleaning, and analysis, using relevant tools and datasets. Educators may require students to design an MSR

study, producing reports that detail the whole set of activities and the rationale thereof required to address specific research objectives.

- *Group projects*. Educators may assign group projects where students can apply MSR techniques to analyze software repositories, like those reported at the end of this chapter, and address specific research questions or challenges. Using group projects, students might jointly design data collection, cleaning, and analysis steps for specific research objectives, having the chance to interact with other students, hence possibly increasing the collective awareness of the most appropriate methodological choices to take, even considering ethical limitations.

2 Setting Up a Source Code Repository Mining Study

This section will overview the MSR process, including the study design, data extraction, and analysis, as illustrated in Fig. 1. We will discuss the considerations for data extraction (e.g., API vs. Package) and storage. Since these steps are typically included to guarantee a systematic mining process, the definition of it might vary depending strictly on the goal of the study that an MSR researcher aims to design. Regarding data analysis, we will focus on aspects that should be considered and errors that could occur about the storage, dataset size, types of analysis, emergent problems, and cases of data contradicting research questions (RQs) or formulated hypotheses.

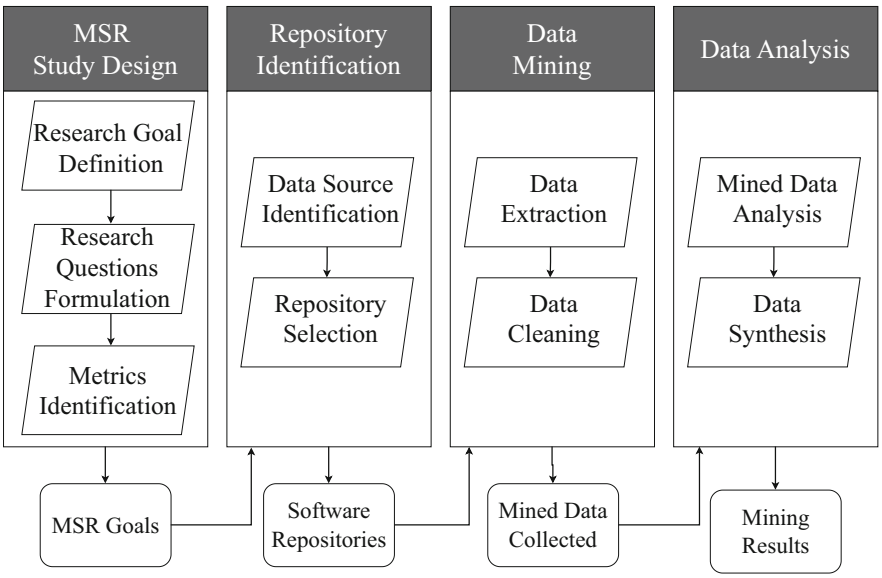


Fig. 1 Set of activities involved in the MSR study process

2.1 MSR Study Design

The first step to designing an MSR study should be systematically defining the research goal. To do so, the widely adopted Goal-Question-Metric (GQM) paradigm of Basili [4] can be used. Designing the objective of a study *via* the GQM paradigm is composed of three main steps, namely, (i) the formulation of the goal of the research, (ii) the design of the RQs that need to be answered to achieve the goal, and (iii) the definition of the metrics necessary to answer the goal.

2.1.1 Research Goal Formulation

The formulation of the research goal of a mining study through the GQM paradigm follows a systematic structure. More specifically, the goal is designed by adopting the following template:

Analyze *the experimental object(s) of the study, e.g., repository source code or GitHub issues*
For the purpose of *the focus of the study, e.g., identifying common bug types or understanding developer behavior*
With respect to *the quality focus, e.g., functional suitability or performance*
From the viewpoint of *the intended reader of the study, e.g., developers or researchers*
In the context of *the context considered, e.g., the Apache ecosystem or Android apps*

On the one hand, formulating the goal according to the template allows one to take the time to reason on the core objectives of the research and start reflecting on the research method to achieve such a goal. On the other hand, reporting the goal by following the GQM template allows one to systematically document and swiftly communicate the research intention to those not conducting the study.

As an example of how a research goal can be formulated according to the GQM template, let us consider the study of Kamei et al. [34]. In the case of this study, a possible formulation of the research goal could be:

Analyze *defect prediction models*
For the purpose of *defect prediction*
With respect to *risky changes*
From the point of view of *software developers and reviewers*
In the context of *open source and commercial projects from multiple domains*

Note that while introducing some systematicity, a single research goal could be correctly formulated in many ways while following the GQM template. Following Occam's razor, the most simple and informative solution should be preferred among different options. Ideally, the research goal is jointly discussed by all people partaking in the mining process. It serves as a collective moment of reflection on the goal and intent of the mining procedure.

A Do for Educators

When formulating research goals, among different options, follow Occam's razor; your students should prefer the most simple and informative solution.

2.1.2 Research Questions Definition

Once the study's goal is defined according to the GQM template (see previous section), the RQs the mining process aims to answer can be formulated. Formulating the RQs based on the research goal allows for further design, with a systematic step-by-step process, of the mining to be executed. RQs should be directly derived from the research goal formulated in the previous step. Once an answer to each RQ is provided, it is possible to assess the extent to which the research goal is achieved (and, in negative cases, further enhance the research process by adding further mining processes or RQs). If needed, during a preliminary repository mining design phase, RQs can be adapted to fit the refined research goal and envisioned repository mining process.

As noted in a recent work by Storey et al. [56] when conducting MSR studies, RQs are often formulated in a rushed manner. To avoid this, a deliberate approach to infer the most suited RQs is recommended. This involves considering the potential impact of the research, drawing inspiration from various sources, and intersecting these with possible phenomena and concepts related to those phenomena. The next step is to brainstorm RQs. This is a creative process where no idea is initially dismissed. The aim is to generate a wide range of potential research questions that could be studied. Once a comprehensive list of potential research questions has been generated, the final step is to select specific research questions for further study. This selection should be justified based on the relevance of the question to the research objectives, the feasibility of answering the question, and the potential contribution of the findings to the field of study.

As a rule of thumb, a mining process is steered by two or three RQs. Having more than four or five RQs usually suggests that the research goal is not well defined or that the RQs are too low level. By taking into account the exemplary study of Kamei et al. [34], we can see how their research goal is decomposed into three different research questions, namely:

RQ_1: How well can we predict defect-inducing changes?
RQ_2: Does prioritizing changes based on predicted risk reduce review effort?
RQ_3: What are the major characteristics of defect-inducing changes?

By comparing the RQs of Kamei et al. [34] with the research goal formulated in Sect. 2.1.1, we can observe that RQ_1 is the one more closely related to the goal. At the same time, RQ_2 and RQ_3 are utilized to build upon the research goal and provide further complementary notions on the considered topic. While slightly striving away from the standard application of the GQM approach, such a technique

can be used to build upon and strengthen the results of the mining process set by the research goal.

As potential inspiration, examples of further RQs of an MSR study could be:
RQ: *Can large language models be used to identify bugs?*
RQ: *Is there a relation between technical debt and software energy efficiency?*
RQ: *How does software quality evolve in microservice architectures?*
RQ: *Which code smells are more frequent in AI-centric software projects?*
RQ: *What are the most common causes of test flakiness?*

From a documentation point of view, once the mining results are collected and analyzed, it is considered a good practice to explicitly answer each RQ (and possible sub-RQs if present) in the mining process documentation. Explicit answers to the RQs are usually documented either in the Results section, Discussion section, or a section dedicated entirely to answering the RQs.

A Do for Educators

A mining process is usually steered by two or three RQs. Usually, having more RQs suggests that the research goal needs to be better defined or that the RQs need to be more high level.

2.1.3 Metrics Identification

As the last step of the GQM approach, each RQ is mapped to a specific set of metrics that do not overlap with those used to answer other RQs. However, this is not always the case. For example, the number of GitHub issues could be used to answer both RQs regarding developer behavior and recurrent development impediments. The selection of metrics strictly depends on the specific RQs at hand, which can be qualitative, quantitative, or a mix of both. Selected metrics can be either atomic, e.g., lines of code or number of defects, or composite, e.g., defects per line of code. It is, therefore, possible to compose the same set of atomic metrics in different manners to answer different RQs.

A Do for Educators

When documenting the metrics used to answer the RQs, students should report their definitions and mappings to the RQs, supporting the metric definition with a sound reference or an unequivocal definition and measurement procedure description.

Teaching Exercises

Through the GQM approach, define a goal, formulate three RQs, and identify five metrics (mapped to the RQs) to investigate which development factors affect bug-proneness.

2.2 Identifying High-Quality Source Code Repositories

2.2.1 Data Sources

The MSR field collects and analyzes rich data from different types of repositories. The repositories can be of static and dynamic nature and can be classified as follows [28]:

- Historical repositories, including (i) Bug or defect repositories, such as Jira and Bugzilla, which help track and manage defects of a software system. (ii) Archived communications such as mailing lists, emails, and chat messages, which are a source of discussions regarding multiple aspects of a software system. (iii) Source control repositories that track all the changes made to software system artifacts (e.g., source code, Pull Requests (PRs), commit messages, documentation). Git is one of the most popular source control repositories.
- Source Code repositories control and manage software projects. Typical features include source code hosting, bug tracking, documentation facilities, mailing lists, and forums. Notable source code repositories include Sourceforge, GitHub, and Bitbucket.
- Runtime repositories such as deployment logs that track the information and actions related to the deployment of a software system. Deployment logs are essential to oversee the configurations and steps of deployment instances. They usually include timestamps, error messages, and configuration settings.

For this chapter, we focused on source code repository mining to align with the running exemplar.

2.2.2 Source Code Repositories Selection Criteria

Code repositories have multiple characteristics [33] that can help in the decision-making process for selection. These characteristics can be used as inclusion and exclusion criteria for narrowing down the candidate repositories.

- Programming language(s), e.g., Java for the Apache Ecosystem projects.
- Size and complexity of the repository, e.g., number of lines of code (LOC) or number of commits
- Domain, e.g., gaming, visualization, database, parsers, and testing.

- Active or inactive repository, e.g., were there recent (during the last 6 months) commit activities on the repository? When was the last commit?
- Types of repositories, i.e., base (not forked) or forked repositories.
- Purpose of the repository. Not all repositories are software repositories. Some are used for experimentation, Web site hosting, academic, and personal (not involving collaboration) projects.
- Location of the repository. Some projects are hosted on multiple platforms.
- Popularity of the repository, e.g., number of stars and contributors.

2.2.3 Considerations when Selecting Data Sources

Several considerations must be undertaken to mitigate potential threats when assessing source code repositories for research purposes. The repositories encompass diverse entities beyond software projects, serving as repositories for free Web storage, online books, or housing projects with other characteristics. Numerous repositories exhibit transient traits, being short-lived, inactive, or dedicated to assignments, student endeavors, educational objectives, personal use, or archival purposes [33]. Thus, if the research question is related to software development, these projects should be removed from the dataset. Furthermore, it is crucial to prioritize repositories that house engineered software projects [42]. Consequently, in studies related to software development, it becomes imperative to purify the dataset for such repositories. Munaiah et al. [42] propose a framework to filter the engineered software projects from GitHub. Additionally, many projects have few commits, and not all use pull requests, resulting in skewed or imbalanced data. The process of commits in pull requests and for code reviews depends on GitHub's practice of recording the commits, which should also be studied carefully.

A strategy to mitigate such potential threats involves refraining from considering repositories that include many non-registered users as committers. Furthermore, projects explicitly identifying themselves as mirrors in their descriptions should be carefully vetted or excluded. For an MSR study, we recommend having a list of inclusion and exclusion criteria based on study goals and RQs. Manual exploration can be conducted on sample repositories to assess whether any changes to the selection criteria are required to ensure a high-quality dataset is collected.

When selecting the repositories, researchers should note that selecting and ranking the repositories based on the number of stars might favor active marketing strategies and not the well-established software engineering practices [6]. Therefore, checking that the repositories are not starred in a short period due to social media activity is essential. It is also important to note that the number of stars is not strongly correlated with contributors, forks, commits, and the repository's age. Thus, relying only on the number of stars can threaten the validity of the collected data.

Another consideration is the correctness of the heuristics and key terms used for selecting the repositories. These heuristics should be scrutinized and documented [28]. The criteria for choosing the data sources may lead to a noisy

collection of repositories and, therefore, a skewed dataset. Careful investigations reveal whether this skewness is related to the inaccurate metrics for selecting repositories or is due to the nature of the data.

Understanding the limitations of repository data is another consideration, as the repository data cannot lead to causal conclusions and only can show correlations [28]. Moreover, the active projects might not include all their development activities in GitHub. So, other resources should be considered.

In general, MSR findings must be considered in the context in which the studies are performed, which is crucial to revealing the actual cause of particular conclusions. Indeed, such findings may not generalize across projects, and repository use could vary between projects. So, researchers should closely examine socio-technical aspects [31] to better understand the use of repositories before reaching conclusions.

When presenting the results of MSR research, the limitations of repository data should be thoroughly examined and communicated. This practice is essential to prevent misinterpretations and ensure research integrity.

A Do for Educators

Students should carefully select the sources before analyzing them. When selecting data sources, the above-stated considerations allow for effective mining, which is essential to answer the RQs, prevent misinterpretations, and ensure research integrity.

Teaching Exercises

A typical exercise would be to select three GitHub projects by focusing on three inclusion and exclusion criteria each, e.g., projects written in different programming languages, of different domains, and active projects with at least 200 commits over the last year as inclusion criteria. Exclusion criteria could include academic and personal projects and projects hosted on multiple platforms. Once the projects have been shortlisted, the issue-tracking data can be mined for further analysis.

2.3 Data Extraction

Data extraction starts once a source code repository is selected and depends on the goals of the mining campaign. In studies concerning predictive analytics, this phase entails mining data concerning independent and dependent variables. The goal is to extract data concerning the former variable able to accurately predict the former variable in the future. For example, let us consider just-in-time defect prediction [34], which aims to predict defective commits given their features and

involves mining commit data to extract valuable information to forecast such defective commits. Such commits can be analyzed sequentially, or specific commits can be analyzed in a given time window.

The dependent variable we will need to extract is the failure proneness of our commits. In contrast, the independent variables are the characteristics that should lead to such failure. It is essential to consider that the described concepts are generalizable to predict defects [34], code smells [2], and security vulnerabilities [66] concerning not only traditional code but also Infrastructure-as-Code [12].

In just-in-time defect prediction, the first step is identifying *defect-fixing* commits. To achieve this goal, we need to collect the issues closed and related to bugs (e.g., with labels `bug` and `bugfix`). Source code repositories like GitHub provide issue trackers that link issue reports and bug-fixing commits. A commit message is tagged as fixing defect if it matches a regular expression like the following:

$$(bug|fix|error|crash|problem|fail|defect|patch)$$

Only the commits that modify at least one source code file are kept.

The second step is to identify the *fixed files*, which are the files modified by *defect-fixing* commits, and their related *defect-inducing* commits.

The commits from the most recent to the oldest are analyzed. For each *fixed file*, the *SZZ* algorithm [52] automatically identifies the *oldest* commit that modified the lines of code involved in the fix. It is worth noting that SZZ has evolved over the years, and several variants are available [49]. Once the defect-inducing commit is found, all commits between the defect-inducing commit (inclusive) and the defect-fixing commit (exclusive) are labeled *failure-prone*.

After the failure-proneness of the code components has been determined, metrics able to predict such failure can be mined. In the example, Kamei et al. [34] mined 14 metrics grouped into five dimensions, i.e., diffusion, size, purpose, history, and experience. It is essential to consider that although some metrics can predict several phenomena (e.g., the file size could lead to defects and bugs), different metrics apply in different contexts. Structural metrics [10] focus on structural properties extracted through source code analysis. Delta metrics [13] capture the amount of change in a file between two successive releases. Process metrics [41] consider aspects concerning the development process rather than the code itself.

In just-in-time defect prediction, all commits could be used to train the models[2]. Nevertheless, analyzing all commits could be infeasible in other contexts; therefore, selection strategies should be applied. For example, only one snapshot for each software release could be analyzed by randomly selecting it or applying another strategy. For instance, the first, last, or middle snapshot of each release could be analyzed.

Finally, several tools are available to mine source code repositories and extract valuable metrics. Among them, PyDriller [53] is a Python framework that helps

[2] In this scenario, the commit order is particularly relevant [19].

developers mining software repositories. PyDriller allows the extraction of information from any Git repository, such as commits, developers, modifications, diffs, and source codes, and quickly exports to CSV files.

A Do for Educators

There is no silver bullet. Although some metrics can predict several phenomena, different metrics apply in different contexts.

A Do for Educators

Do not re-invent the wheel. Several tools are available to mine source code repositories and extract useful metrics. Before developing a brand-new tool for mining software repositories, explore existing ones.

2.3.1 Data Cleaning

Software repositories often contain various types of noise that can skew the results of analyses [3]. Therefore, data cleaning is a critical step in preparing data, which includes a range of techniques and procedures to ensure that datasets are accurate, consistent, and reliable. Several techniques are employed to ensure the adoption of high-quality data before analysis, a crucial practice, particularly in the context of artificial intelligence (AI). Different methods of handling data greatly depend on the specific context in which it will be used.

This section is designed to help readers manage data in the context of MSR, emphasizing one of the most critical aspects of software repositories: the commit. This element can hinder different issues after the data are extracted and can lead to drift in the results of our analysis. Commits are puzzle pieces that make up a project's story. Each commit contributes to the bigger picture, but sometimes, some do not fit perfectly, leaving gaps or distorting the intended image. Addressing these problematic commits is essential to maintaining the integrity of the entire project and ensuring the accuracy of our analyses.

When mining data from commits, the first issue concerns extracting data from *tangled commits* [30]. This type of commit contains multiple pieces of information related to different changes, caused often by the introduction of *multiple-fixing change commits* [43]. These changes during the development phase may not cause major issues. However, they can create problems while analyzing the corresponding version archive by introducing inaccuracies. For example, if a complex change is made to fix a bug, all the files associated with it may be mistakenly labeled as defective in the historical context. When treating these types of commits, it is

suggested to understand the changes inside a commit and consider it separately as *change set partitions* [29].

An additional critical issue that arises during the data collection process is the inclusion of *merge commits*, which has been highlighted as a nuanced issue by Kovalenko et al. [36]. This issue requires an additional effort since many merge operations may not be labeled as such, leading to inspecting the content of the changes manually to identify it [33]. Similar to *merge commits*, it is necessary to consider other types of changes and commits that can mislead the analysis. In this set, we can also find *quickly remedy commits* (i.e., commits aimed at implementing changes omitted in the previous commit) [63]. The decision to include or exclude these commits can significantly impact the complexity of the dataset and the richness of the information obtained. Wen et al. discovered that excluding this type of commit allows us to avoid introducing a significant amount of noise in MSR studies [64]. Incorporating all commits, including those outside the main branches, can elevate the complexity of the analysis. However, it can also enrich the dataset by adding important information to the software history and providing a more comprehensive view of collaborative efforts, branching strategies, and concurrent development threads.

On the other hand, limiting the analysis to the main branch can simplify the extraction of essential information. However, it may also lead to missing valuable insights into the other branches. Therefore, the inclusion or exclusion of these commits presents a delicate balance, and MSR researchers should carefully consider the trade-offs involved. As such, it is crucial to weigh the benefits and drawbacks of each approach, keeping in mind the research goals and the nature of the data under investigation.

When determining whether to include or exclude commits based on the goals of the MSR study, it becomes evident that commits containing information not aligned with the study's objectives are considered noise. *Noisy commits* in software development and version control systems refer to changesets or commits deemed extraneous to the primary focus of analysis. These typically involve minor modifications like fixing typos, adjusting task names, or resolving lint warnings [37]. In MSR studies, researchers often encounter noisy commits and filter them out to improve the signal-to-noise ratio, allowing for a more targeted analysis of substantive changes. Dalla Palma et al. [12], for instance, exemplified this approach by excluding commits that addressed typos, task names, and lint warnings, prioritizing more impactful contributions to the codebase. Additionally, an MSR researcher should consider including or excluding bot commits [16]. Detecting and considering the exclusion of this type of commit can significantly differ on several aspects involved in MSR analysis, including community-related aspects [15]. This commit type is detectable by observing the commit message and the list of changes the bot will perform.

To conclude, an MSR researcher must carefully navigate the decisions surrounding commit inclusion or exclusion, acknowledging the significance of commits in shaping the quality of data analysis. By aligning these decisions with the specific goals of the study, researchers ensure that the dataset remains focused on substantive

contributions, enhancing the precision and relevance of their findings. Therefore, an MSR researcher should take the following steps to obtain the optimal dataset:

- Identifying the criteria for selecting the key commits that should be included in the study.
- Consider including or excluding tangled, merge, and quick-remedy commits.
- Based on the defined criteria, identify and exclude the noisy commits in the extracted data.

A Do for Educators

Students should carefully clean data. Each piece of data is a puzzle piece that makes up the story of an MSR campaign. They all contribute to the bigger picture, but sometimes, some do not fit perfectly, leaving gaps or distorting the intended image.

Teaching Exercises

A practical exercise for understanding the data extraction phase involves students applying algorithms from the study of Kamei et al. [34] to identify bug-inducing and fixing commits in repositories. Starting with chosen repositories, students identify commits that fix defects. During this process, they meticulously clean the data by removing any tangled and merge commits. Finally, students extract the commit message and the set of files modified by each defect-fixing commit.

2.4 Code Analysis

Source code analysis is extracting information about a program from its source code or artifacts (e.g., from Java byte code or execution traces) generated from the source code using automatic tools [5]. Source code analysis can be textual, static, dynamic, and historical [17]. We focus on static code analysis.

2.4.1 Static Code Analysis

Source code is a crucial artifact that can be analyzed to reveal information. A technique to analyze source code is static code analysis (also known as structural analysis)—the process of checking the source code of a program for issues without executing the program. Static code analysis is helpful in (i) ensuring that code adheres to rules about good coding practices, (ii) finding defects and security issues, and (iii) identifying code smells and technical debt instances. Static analysis can be

applied during the early stages of software implementation for early fault detection since the code does not need to be fully functional or executable.

Source code can be analyzed at different levels of granularity. The analysis level depends on the study's goal and research questions, which will, in turn, dictate the level of granularity at which the analysis needs to be performed. Generally, analysis can be performed at the (i) method level, the lowest level of granularity that allows finer-grained analysis; (ii) class level, a coarser level of granularity compared to method level but still popular among researchers; (iii) file level, a higher level of granularity and lower level of details when analyzing groups of classes; and (iv) system level, the highest granularity level and the lowest details level.

Tools such as **Automated Static Analysis Tools (ASATs)** are increasingly used for static analysis. ASATs can be of general purpose (e.g., PMD, SonarQube, Understand, Cppcheck), bug-focused (e.g., SpotBugs, Coverity), and security-focused (e.g., Flawfinder, Fortify). Static analyzers differ based on the level of sensitivity regarding precision and analysis time [18]. A flow-sensitive analysis is more precise but more time-consuming and considers the order of the statements compared to a flow-insensitive analysis. A path-sensitive analysis considers only valid paths and can be precise but costly, while a path-insensitive analysis considers all execution paths, including infeasible ones. Context-sensitive (also known as inter-procedural) analysis considers the context during the analysis compared to an intra-procedural analysis. The latter is faster but more imprecise compared to an inter-procedural analysis.

The Open Worldwide Application Security Project (OWASP)[3] provides a comprehensive list of security static analyzers. Some commonly used static analyzers are presented next.

PMD[4] is a general-purpose static analyzer that supports 16 languages but primarily focuses on Java. For instance, Java rules enforce accepted best practices and coding styles; uncover design issues; detect constructs that are either broken, extremely confusing, or prone to runtime errors; and flag code documentation issues, suboptimal code, potential security flaws, and issues when dealing with multiple threads of execution.

SpotBugs[5] (formerly FindBugs) is a static analyzer mostly focused on defect detection based on a predefined set of more than 400 bug patterns in Java code. These bug patterns check for bad practices, correctness, performance, malicious code, and security issues.

Fortify[6] is one of the most popular static analyzers specifically for violations of security-specific coding rules and guidelines in multiple languages. Fortify comprises eight vulnerability analyzers: buffer, configuration, content, control flow, dataflow, null pointer, semantic, and structural.

[3] https://owasp.org/www-community/Source_Code_Analysis_Tools

[4] https://pmd.github.io/

[5] https://spotbugs.github.io/

[6] https://www.microfocus.com/en-us/cyberres/application-security/static-code-analyzer

2.4.2 Dynamic Program Analysis

Dynamic program analysis involves analyzing the properties of a program while it is executing with real input data. Unlike static analysis, dynamic analysis aims to identify issues while the code runs. Dynamic analysis is useful in identifying (i) a lack of code coverage, (ii) memory allocation and leaks, (iii) performance bottlenecks, (iv) software vulnerabilities and defects, and (v) concurrency issues such as deadlocks and race conditions.

A comprehensive list of dynamic analysis tools can be obtained from the analysis-tools.dev Web site[7]. Below are some notable **dynamic analysis tools**.

Valgrind[8] provides a suite of tools for building dynamic analysis tools. For instance, Memcheck can detect many memory-related errors that are common in C and C++ programs and can lead to crashes and unpredictable behavior, Helgrind is a thread debugger that finds data races in multithreaded programs, and Massif performs detailed heap profiling by detecting which parts of the program are responsible for the most memory allocation.

Application Verifier (AppVerifier)[9] by Microsoft is a runtime verification tool for unmanaged code to detect and help debug memory corruptions, critical security vulnerabilities, and limited user account privilege issues that would be difficult to detect during regular application testing.

Code Pulse[10] is a real-time code coverage tool for penetration testing activities by OWASP and Code Dx (acquired by Synopsys[11]) and automatically detects coverage information while the tests are being conducted. Code Pulse currently supports Java and .NET Framework programs.

Teaching Exercises

A useful exercise for understanding the utility of performing static or dynamic analysis on code is to analyze the impact on code quality before and after a defect-fixing commit. Students will retrieve versions of the code from a specific repository both before and after a defect-fixing commit. By utilizing a chosen analysis tool, such as SpotBugs, students will explore the implications of defect fixes regarding correctness, performance, and security issues.

[7] https://github.com/analysis-tools-dev/dynamic-analysis

[8] https://valgrind.org/

[9] https://learn.microsoft.com/en-us/windows-hardware/drivers/devtest/application-verifier

[10] https://code-pulse.com/

[11] https://www.synopsys.com/software-integrity.html

2.5 Mined Data Analysis

The analysis of the collected data depends mainly on the data type but also on other factors such as the purpose of the analysis and the tools and techniques available. Even when source code repositories are analyzed, there are multiple data sources and types: source code, commits, GitHub issues, questions and answers, review comments, code comments, numerical data, etc. In this section, we will review some data types and techniques for analyzing the mined data. As we have focused on code in the previous section, we cover a broader range of artifacts in this section. Although source code analysis requires specific techniques, which are beyond the scope of the current chapter, the topic modeling and deep learning-based approaches mentioned in this chapter are widely used for analyzing the source code.

2.5.1 Unstructured Data

"Unstructured data does not have clear, semantically overt, easy-for-a-computer structure. It is the opposite of structured data, the canonical example of which is a relational database, of the sort companies usually use to maintain product inventories and personnel records." [9]

Textual data such as commits or source code are unstructured, as they do not have a pre-defined format. Instead, the authors can write the text in any order and with different contexts, even though a template is used. Understanding the content of the text is of main interest in many studies and requires analysis. We can use several techniques to analyze textual data. The main factor of using the analysis technique is "what" we intend to identify in our study. This can vary from more straightforward techniques, such as analyzing the frequency of words based on Term-Frequency Inverse-Document-Frequency (TF-IDF) metrics, to more in-depth analysis to understand the content of the documents or categorize them, such as topic modeling, clustering, and classification techniques to sophisticated analysis such as finding the reasons, relations, or causality among concepts, using more advanced text analysis techniques such as causal inference. In all cases, the textual data could be represented with numerical vectors (also referred to as vector embedding) that can be learned using deep neural models.

Topic Modeling

Topic modeling or latent topic modeling is a category of techniques for automatically extracting topics from a corpus of text documents. The corpus of documents can be short or long texts, such as commit messages, app review feedback, or bug reports. Please note that each commit, app review, or bug report is considered a document in this example. A topic is a collection of terms that frequently occur in the documents. When topic modeling is applied, the main topics of the documents'

corpus are extracted; therefore, we can compute the topics of each document as well. Thus, topic modeling uncovers the latent semantic relationships of the documents. As this ensures a faster analysis of many documents, topic modeling is used for several software engineering applications such as bug triaging, finding the most discussed topics or issues, traceability link recovery, and concept location. Multiple applications of topic modeling in software engineering research are discussed in the literature [51].

Examples of topic modeling algorithms are Latent Semantic Indexing (LSI), Probabilistic Latent Semantic Indexing (PLSI), and Latent Dirichlet Allocation (LDA) and its variations, such as Hierarchical Topic Models (HLDA), non-negative matrix factorization (NMF), and Biterm Topic Model (BTM). While LDA and other algorithms are designed for longer text, BTM is a newer algorithm for short text. The BTM algorithm considers the Biterms in the whole corpus to enhance the topic learning in short texts. Online algorithms for BTM, i.e., online BTM (oBTM) and incremental BTM (iBTM), are also introduced to speed up the inference of BTM on large data sets.

The algorithms mentioned above are mainly used to extract the topics from documents at a static point in time. As the distribution of the terms in a document changes over time, so can its topical topic modeling techniques, which are developed to detect the evolution or variations of topics in time-stamped documents. Dynamic Topic Model is one of these algorithms. Online Latent Dirichlet Allocation (OLDA) is another method that tracks the variations of topics over text streams. The OLDA models the texts' topics of one time slice based on the topics of the last time slice. Newer algorithms are developed that consider previous time slices of the documents for better topic modeling. Adaptively Online Latent Dirichlet Allocation (AOLDA) is an algorithm specifically developed for software engineering and app review analysis. It improves OLDA by adaptively combining the topic distributions of previous versions to extract topics in the current time slice. Adaptive Online Biterm Topic Modeling (AOBTM) is a similar algorithm developed for short texts. It analyzes the statistical data of previous time slices to identify the topic distribution of the current time slice.

The more recent topic modeling algorithms are based on word embeddings or neural network-based topic models, such as Top2Vec, LDA2Vec, and BERTopic.

Considerations. When applying topic modeling, several considerations are necessary to improve the results:

- Text pre-processing: Though the text pre-processing varies based on the study context and data, the common techniques are to remove punctuation and non-textual symbols, check the language of the text (e.g., English only), spell-check and correct the spellings of the words, stemming and lemmatization, and removing stop words.
- Investigating the length of the text: It is shown that the topic modeling algorithms do not work correctly for short text. So, investigating the length of the documents is necessary in selecting the topic modeling algorithms, whether they are designed for short or long text.

- The evaluation metrics: The evaluation metrics of topic models span over quality, interpretability, stability, diversity, efficiency, and flexibility. Two well-known terms to measure the topic models are perplexity and coherence. The former refers to how well the model explains the data (predictive power of the model), and the latter identifies a measure of whether the generated words for a topic can be associated with a single semantic meaning. One way to evaluate coherence is Pointwise Mutual Information (PMI). For other evaluation metrics, the work of Abdelrazek et al. [1] provides a good starting point.
- Number of topics: The user should choose the number of topics in most algorithms. Rather than using a random number or based on the heuristics, a common way is to explore a range of different topic numbers and evaluate the coherence score of the results, which could be a good determination to select the number of topics. There are other metrics for models such as BERTopic that should be evaluated for selecting the best number of topics.
- Hyperparameter setting: Similar to the previous point, setting the hyperparameters is important in running the algorithms. So, care should be taken to experiment with different choices before applying the topic modeling.
- Running time and size of data: One main consideration in choosing the topic modeling algorithm is the time it takes to execute the algorithm and whether it can be applied to a large dataset.

Sentiment Analysis

Sentiment analysis, also referred to as opinion mining, refers to gathering and analyzing people's opinions, emotions, or attitudes toward an entity (i.e., individuals, product, topic, etc.) [62]. In software engineering, sentiment analysis can analyze various MSR artifacts, such as app reviews, users' product feedback, and developers' discussions. Sentiment analysis determines whether the opinion is positive, negative, or neutral. The analysis can be done at different levels of granularity: document, sentence, phrase, or even certain aspects of an entity. Several algorithms are developed for sentiment analysis, from lexicon-based approaches to unsupervised and supervised machine learning techniques, including deep learning models and pre-trained language models, hybrid approaches, and transfer learning approaches.

Considerations While for some text analysis applications, the text should be cleaned and punctuation marks removed, text pre-processing considerations for sentiment analysis should be done carefully. Several essential features can reveal sentiments, including punctuation marks, emojis, and slang words, which are some selected features for sentiment analysis.

(Deep) Neural Network Based

Another common technique for analyzing textual data is representing the document with a vector. This vector, called embedding, can be a non-contextual or contextual text representation. For non-contextual representations, the model learns a fixed vector for each word, no matter their context.

Famous examples of such embeddings are word2vec, glove, and code2vec. In the contextual representation, the word embedding depends on its surrounding words. Contextual representation has recently been used in many applications in which the embedding of the words or text is extracted from a language model, a model that learns a probabilistic model of a natural language from a large corpus of textual data. Multiple models can be used to extract the embeddings of text or code, including Sentence Transformers or GPT family for text or code-specific language models such as CodeBERT, GraphCodeBERT, CodeT5, or Code Llama. Many newer models can also be used, and we refrain from naming them due to the increasingly fast introduction of newer models. While topic modeling and sentiment analysis are techniques aiming at a specific goal (i.e., understanding what is being discussed and what emotion is expressed), deep learning models are generally discussed and can be used for various purposes, including topic modeling and sentiment analysis. These models or embeddings extracted from them can be applied in other analyses such as predictive analysis or training machine learning models. Some examples of the studies that use deep learning for mining software repositories are mentioned in the survey by Yang et al. [68].

Considerations In the following, we list a few points to consider when choosing the models.

- Cost of using a model. Most of the recent language models are based on deep learning approaches. The computational costs available to the researchers should be considered when using them. Additionally, if using models such as a GPT-based family, one may intend to pay for the calls to the model and receive the embeddings. Therefore, the budget associated with the research based on the dataset size is an important factor.
- Model selection. Several non-contextual and contextual models are specifically developed for software engineering tasks and code. Depending on the application, we recommend a search to identify the model to use in the analysis. Several SE-specific language models have been developed, such as seBERT, which is a BERT model trained from scratch on software engineering data. These specific models are available for various applications and domains in software engineering. It is necessary to conduct at least some initial studies if there is insufficient literature to ensure the final model is chosen properly.

2.5.2 Structured Data

Structured data are data that have a standard format. Examples of structured data are time stamps, number of commits, and years of experience. Structured data are analyzed through various means, such as correlation analysis, time series analysis, statistical analysis, clustering, regression analysis, and developing prediction models. Note that some techniques, such as clustering, and developing machine and deep learning models might apply to the unstructured data. A discussion about machine learning strategies for MSR can be found in [26]. Both structured and unstructured data can be used for the analysis or extraction of quantitative metrics while working with unstructured data. For example, the mean number of accepted answers on Stack Overflow can be calculated while also analyzing the contents of the questions and answers.

In the following, we briefly describe some of the statistical tests. Statistical methods, or models, are powerful tools for analyzing data and supporting arguments. They are mathematical formulas used to analyze numerical raw data. The study conducted by De Oliveira Neto et al. [14] provides a survey on different tests that are used for empirical studies in software engineering, which can be used as a starting point to review the widely used statistics tests. Parametric and non-parametric tests are specific classes that refer to the distribution of the population's parameters. The t-test, ANOVA, and F-test are some common parametric tests. Examples of non-parametric tests are Mann-Whitney U test and its variations, Kruskal-Wallis test and the χ^2 test. If multiple tests are conducted, correction tests such as Bonferroni or its variations should be considered.

Another method is statistical power analysis is a test conducted to accept or reject a hypothesis. The statistical power refers to the probability of a hypothesis finding an effect if there is an effect. Power analysis uses a significance level, effect size, and statistical power to estimate the required minimum sample size.

Considerations In the following, we list a few points to consider when analyzing structured data.

- Data distribution. When applying statistical tests, it is imperative to assess data distribution. Statistical tests are often developed for normal or non-Gaussian distributions, and one cannot be applied to the other. Therefore, the results are unreliable if the data follows a non-Gaussian distribution, but the test is for normal distributions. Shapiro-Wilk and Kolmogorov-Smirnov tests are commonly used to assess the normality distribution of datasets.
- Reporting effect size. When applying statistical tests, it is necessary to understand and reflect on the effect size and significance level and not choose them arbitrarily. When reporting the results of a statistical significance test, usually p-values are reported but not the effect size. However, a p-value only informs the existence of an effect but not its magnitude (i.e., effect size). Therefore, the effect size should be reported to find out the size of a significance [57].
- Often, to develop an analysis, the model's features combine structured and unstructured data. In this case, the embeddings of the text or code could be used

as features, or the text itself could be considered. There are several approaches to be used, and one can experiment with different combinations to evaluate the results.

A Do for Educators

Data analysis depends on the data types. Even for source code repositories, there are multiple data sources and types. Tools must be selected considering the nature of the repositories and the analysis to conduct.

Teaching Exercises

A useful exercise for understanding data analysis in MSR involves performing topic modeling on defect-fixing and non-defect-inducing commits. Students will start with a set of commits from Kamei et al. [34] and preprocess the commit messages to ensure data quality. Students will identify the main topics discussed in defect-fixing and non-defect commits using a specified topic modeling technique, such as Latent Dirichlet Allocation (LDA) or BERTopic. They will analyze and interpret these topics, comparing the topics of defective and non-defective commits and exploring their relevance to different types of defects and their implications for software maintenance practices. A similar practice can be applied using sentiment analysis to investigate whether the sentiments of commits change when fixing a defect compared to a non-defect one.

2.6 Data Synthesis

Collecting and synthesizing the analysis results is crucial to interpreting data and discovering important findings. Before presenting all the results extracted from data analysis, an MSR researcher's primary focus is to remember the goal addressed in the previous steps. Following the GQM approach (presented in Sect. 2.1), all the plots and the results shown in the study aim to answer the research questions formulated. Therefore, before starting, it is crucial to keep the following points in mind:

- **Research Objectives as a Guide:** Using the formulated research goals helps organize and prioritize your results. Therefore, considering a mapping link between the research question and the findings helps find the best solutions to present results through discussion, plots, and tables.
- **Relevance to Research Questions:** Articulating how each result addresses specific research questions helps check if the goal is accomplished. This approach

ensures that the presentation of the results maintains a cohesive narrative and directly ties back to the study's purpose.
- **Embracing Unexpected Findings:** During an MSR study, it is possible that the results show something different from what is expected. The presentation and discussion of unexpected results could enlighten new implications and open the possibility of novel opportunities.

Once the goal is clearly assessed, it is necessary to format the data and the outcome of the analysis to obtain the information needed to answer our research questions. Therefore, summarizing and exploring the data is important to extract the veiled message from large collected data.

The main approaches to summarize the analysis results are based on descriptive metrics and visual representations.

2.6.1 Descriptive Metrics

Using properly descriptive metrics and presenting information is integral to conveying the study results. In summarizing metrics derived from MSR, descriptive statistics are pivotal in distilling complex datasets into meaningful and interpretable insights. The choice of descriptive statistics should be adopted, focusing on the goals of the analysis to find the most suitable answers from the data. Key descriptive statistics capture the central tendency, variability, and distribution of the metrics, including mean, median, mode, and standard deviation [35]. For instance, the mean serves as a central measure, offering an average value that summarizes the overall trend within a dataset. However, it is important to note that the mean can be influenced by extreme values, making it less representative of skewed distributions. In the context of software components, calculating the mean number of lines of code provides an overview of the project's codebase size, but it may not fully capture the central tendency if the distribution is highly skewed. The median, resistant to extreme values, provides a robust representation of the center of the data. In MSR, identifying the median response time for resolving issues can offer a more stable representation of the typical resolution time, even in the presence of outliers. Mode highlights the most frequently occurring values, emphasizing prevalent patterns. For instance, identifying the mode in a dataset of commit frequencies may highlight the most common development activity intervals. Standard deviation quantifies the dispersion of data points, offering insights into the dataset's variability. For example, examining the standard deviation of code churn rates can reveal the extent to which development activity is variable across different periods. Collectively, the combination of several descriptive statistics contributes to a comprehensive understanding of the metrics, aiding researchers and stakeholders in uncovering patterns, trends, and central characteristics within the intricate landscape of software development data.

2.6.2 Visual Representations

Descriptive statistics offer a concise and straightforward way to present data analysis outcomes. However, relying solely on numerical summaries, such as means and medians, can obscure significant patterns and variations within data. This approach may also fail to capture the complexity of relationships or variations in multidimensional data. To overcome these shortcomings and improve the interpretability of the results, it is crucial to supplement numerical summaries with visual representations. Visualizations provide an interactive and intuitive method for discovering patterns, outliers, and correlations within the data. Employing various visual elements such as plots, tables, and representative graphs when synthesizing your data will increase the comprehension of the readers interested in the study and allow them to catch it.

As for the descriptive metrics, each plot type serves a unique purpose in conveying information effectively. Some of the most used visual representations to summarize the results in MSR studies are:

- **Line charts**: This type of plot is helpful to visualize temporal aspects of software repositories, showcasing the evolution or the history of metrics over time. An example of this type of plot for MSR Studies can be related to analyzing the use of third-party libraries over time [50].
- **Bar plots**: These plots show the frequency of a categorical variable using bars. The bars' height indicates the data's value in each category, such as the frequency, total count, sum, or average. An example of using bar plots for MSR studies is to analyze the number of occurrences of bug categories in software systems [7].
- **Box plots**: These plots show the distribution of a numerical variable using five statistics: the minimum, the lower quartile, the median, the upper quartile, and the maximum. Box plots can be used to compare the central tendency and the variability of different groups or samples of data. An example of using box plots for MSR studies can be related to the extraction of the number of commits that lead to the appearance of a code smell [60].
- **Scatterplots**: This type of plot is used to show the relationship between two numerical variables. The position of each dot on the horizontal and vertical axis indicates the values of the variables for each observation or unit of analysis. Scatterplots help explore the correlation or association between two variables or identify outliers or clusters in the data. Scatterplots are commonly used to analyze a specific factor's evolution to find tendencies. An example of using scatterplots is analyzing the adoption of reusability mechanisms in source code over time to find common patterns between projects [22].
- **Network graphs**: This type of plot is commonly used to model relationships between elements. It can be used to visualize clusters and patterns in the data and analyze complex systems' properties and behaviors. Network graphs in MSR studies are commonly used to explore collaborative networks between developers [27].

2.6.3 Best Practices and Dos for Educators

The following best practices are collected to inform MSR researchers of the main requirements for reporting an MSR study.

Replicability

In MSR studies, it is crucial to document data sources, collection methods, preprocessing steps, and analysis techniques for easy replication by other researchers. MSR researchers should always provide all the material to reproduce the conducted studies, including code, scripts, or workflows. All this content should be in an accessible collection called *replication package*. Mahmood et al. [38] proposed using common online replication services, such as OpenML[12] and Zenodo[13], to replicate MSR studies effectively.

Use of Visual Elements

When incorporating visual elements in information representation, prioritize clarity, consistency, accessibility, and relevance. Visualizations should be designed with simplicity to ensure easy comprehension for a diverse audience. Consistent use of visual elements fosters a cohesive visual language, while accessibility features, such as color-blind-friendly palettes and alternative representations, promote inclusivity. Additionally, colors can be strategically used to represent additional information, adding depth to the visualization. Visualizations should serve a clear purpose and contribute meaningfully to the overall narrative, enhancing the interpretability of complex data.

Use of Relative Discussions to Findings

When presenting research findings, it is important to emphasize their significance and relevance to the research questions. Researchers should discuss existing literature, industry benchmarks, or expectations to comprehensively understand the relative importance of the findings. By highlighting the connections between the findings and the external contexts discussed, researchers can offer richer insights and demonstrate the practical implications of their work. This approach ensures that the reported results are not isolated but instead contribute meaningfully to the ongoing goal of the study.

[12] http://www.openml.org/

[13] https://zenodo.org/

Use of a Systematic Quality Assessment Process

When all research components are in place, employing a systematic quality assessment process is prudent. This systematic approach ensures a thorough and rigorous evaluation of all artifacts generated throughout the study. The assessment process should encompass a comprehensive review of the dataset, code, visualizations, and other materials, verifying their accuracy, completeness, and adherence to established standards. MSR researchers can identify and rectify potential errors or inconsistencies before finalizing the study's outcomes. To have an effective systematic evaluation process of the MSR study, Chatterjee et al. [8] proposed a series of standards that an MSR researcher should use to ensure high-quality artifacts.

Teaching Exercises

A good exercise is reproducing an existing study, including related synthesis, analysis, and interpretation of the results. Considering the outcome of the previous exercise that highlights the most used topics in defect-fixing commits, students will attempt to synthesize the data, collecting key elements and interesting findings. They will create plots to observe critical characteristics of the synthesized data and make interpretations of the results to explore the topics in defect-fixing commits.

2.7 Threats to Validity in MSR Studies

While often considered a mere afterthought of mining processes [61], threats to validity (TTV) play an essential role in empirical inquiries. Summarily, threats to validity could potentially affect the accuracy or credibility of a study or its results [65]. A transparent, comprehensive, and truthful documentation of the TTVs that may have influenced a study is essential. TTV considerations are crucial to let the reader accurately understand the mining process, interpret its results, and potentially build upon them. TTVs and related mitigation strategies should be considered throughout the entirety of the mining processes, starting from the earliest stages (e.g., by reflecting if the RQ is correctly formulated to achieve the research goal) till the concluding steps. Among different TTV categorizations, often MSR studies follow the four categories presented by Wohlin et al. [65], namely, *conclusion validity*, *internal validity*, *construct validity*, and *external validity*.

2.7.1 Conclusion Validity

Threats to the conclusion validity refer to impediments that may affect the ability to draw the correct conclusion about relations in the collected data. Leaving

aside conclusion threats related to statistical result analyses (e.g., low statistical power and violated statistical test assumptions), recurrent conclusion threats in MSR processes regard confounding factors, such as measure reliability (mainly if dynamic code analyses are used), random irrelevancies in the mining process (e.g., including extended periods of repository inactivity), and excessive heterogeneity of repositories (e.g., considering in the same MSR process both cloud-native and embedded contexts). Mitigation strategies for conclusion TTVs in MSR studies often entail the selection of repositories based on *a priori* defined criteria, the carefully motivated use of sound source code analysis tools, and *post hoc* scrutiny of results trends to spot potential conclusion pitfalls related to specific repositories.

2.7.2 Internal Validity

Threats to the internal validity regard unknown factors that can impact the study's relationship between the phenomenon considered and the observed results. Threats of this nature are often related in MSR processes with a summary repository selection, inaccurate mined data post-processing, unfitted statistical data analyses, and, when dynamic analyses are used, the influence of previous code executions on subsequent ones. Internal TTVs of MSR studies can often be tackled during the mining design phase by ensuring that a systematic, repeatable, and documented process is used to select repositories and manage the collected data, and "cooldown" and cache cleaning precautions are taken to start each dynamic measurement as a clean slate.

2.7.3 Construct Validity

Construct validity pertains to the representativeness of the designed mining process to accurately study the considered theoretical construct. Most commonly, construct threats in MSR processes relate to the design phase of the mining processes and may involve a vague or ill-suited definition of the construct (e.g., defining technical debt as maintainability issues), underrepresentation of the considered construct (e.g., considering a single type of test flakiness to study the topic), or the adoption of an overly narrow analysis to examine a considerably broader construct (e.g., focusing solely on number of functions to study software testability). During the design of MSR processes, the construct under study and how the mining considers such construct should be carefully discussed among miners to understand, mitigate, and document the potential construct TTVs entailed by the adopted methodology.

2.7.4 External Validity

External validity regards the extent to which the results obtained with the mining process can be translated into other contexts (e.g., industrial practice, another application domain, or a different programming language).

Among the external TTVs, one of the most common ones in MSR regards the underrepresentation of the entire set of software repositories and commits relevant to answering the RQs. A strategy to systematically sample repositories and commits is paramount to mitigate this threat by considering the threat such sampling entails. A sound sampling should carefully select both a fitting sampling technique (e.g., stratified sampling if repositories of different natures need to be represented) and a sample size. Regarding sample size, numerous tools can be used to calculate it based on confidence interval and margin of error, typically 95% and 5%, respectively, but these can vary according to the entire set of repositories considered.

Another common external TTV of MSR studies is adopting *ad hoc*, manual, or outdated processes to mine and analyze the source code. Conscious attention should be paid to ensure that systematic and repeatable state-of-the-art and practice processes are used, i.e., analysis tools and considered repositories, to mitigate this threat.

Additionally, depending on the data under study, other sampling techniques, such as stratified sampling, should be considered to ensure that the distribution of the sample data in each class or group is retained and similar to the distribution of the classes in the original data.

3 Complementing Software Repository Mining Studies

MSR studies might be combined by analyzing additional sources of information. From a methodological standpoint, this is the basis of *mixed-method* research [11, 55], a research approach combining qualitative and quantitative research methods elements within a single study or research program. This approach seeks to harness the strengths of both methodologies to provide a more comprehensive and nuanced understanding of a research question or phenomenon. Storey et al. [55] recently defined guidelines aiming to ease the application of mixed-method research in software engineering, as well as a catalog of best and bad practices to help apply it. In the scope of MSR research, mixed-method research may be useful to enhance or confirm the findings coming from mining human-generated data but also enable asking different questions as they arise during the study [55]. In doing so, researchers may triangulate the findings by exploiting multiple research methods, increase the overall credibility of the findings, or even find contradictory or surprising results [55].

In the following section, we overview the potential methodologies that may be used to complement the results of MSR studies and provide exemplary articles that may be used to illustrate these methodologies in practice.

Complementing MSR Research with Qualitative Methods For instance, imagine complementing an MSR exploration with qualitative research methods, such as *surveys*, *interview studies*, and *focus groups*. A survey represents a research method that involves collecting data from a sample of individuals by administering a structured set of questions. When combined with MSR studies, a survey could

capture the subjective experiences and perceptions of developers, project managers, and other stakeholders, providing a human narrative to complement the quantitative trends identified in the MSR exploration. Literature has often relied on this combination of research methods. A notable example is the paper by Qiu et al. [46], where the authors effectively combined software repository analyses and insights from a survey study to investigate the impact of social capital on the sustainability of open-source projects.

Interviews represent an alternative to survey studies. An interview is a qualitative research method that systematically collects and analyzes data from one-on-one interviews with participants. Unlike survey studies, interviews provide researchers with finer-grained insights from the interviewees' experiences. By nature, interview studies can only reach a small sample size and are typically limited to the analysis of a few practitioners. As a consequence, interview studies are particularly suitable when the aim is to gain a deep and nuanced understanding of participants' experiences, perspectives, or beliefs. For example, interview studies with key stakeholders offer a deeper understanding of individual experiences, motivations, and decision-making processes, adding a layer of context to the automated data. On the contrary, if the goal is generalizability, there might be better research instruments than an interview study. The interested reader might take the paper by Tao et al. [58] as a valuable example of how to make interviews instrumental for the goals of an MSR exploration.

Focus groups represent an additional alternative. These refer to a qualitative research method that involves a small, diverse group of participants who engage in an open and facilitated discussion about a specific topic under the guidance of a moderator. This method aims to gather insights, perceptions, opinions, and attitudes through group interaction, allowing participants to express their views and respond to each other in a dynamic setting. Like interview studies, focus groups rarely have the power to generalize the insights that emerge and should be used to understand better the quantitative findings obtained by an MSR study. An example of applying this combination can be found in the paper by Falessi et al. [20].

Complementing MSR Research with Quantitative Methods Besides qualitative research methods, it is also worth reporting that an MSR study can be empowered using additional mining instruments beyond the scope of the traditional version control systems. For instance, *issue trackers*, where bugs and tasks are recorded and discussed, become a valuable trove of information. Mining these repositories unveils the challenges developers face, the evolution of bug resolution processes, and the collaborative dynamics surrounding issue resolution. At the same time, *code review repositories*, theaters where code changes are scrutinized, are an opportunity to explore the quality assurance practices within a project. By analyzing discussions, comments, and decisions made during code reviews, researchers gain insights into coding standards, knowledge transfer, and the social dimensions of code evaluation. Finally, *developer forums* like Stack Overflow become digital arenas where practitioners seek and provide solutions. Mining these forums provides a glimpse into the knowledge-sharing ecosystem, exposing common challenges

developers face and the collaborative solutions the community offers. A notable example of the combination of multiple mining instruments can be found in the work by Ram et al. [47].

Key Advantages of Mixed-Method Research In the scope of MSR, integrating qualitative research methods and exploring additional mining instruments offer a nuanced perspective. While MSR studies provide the backbone of empirical evidence, qualitative methods infuse a *human dimension*, unraveling the stories behind the code changes. Delving into issue trackers, code review repositories, and developer forums adds layers of context, portraying software development as *a collaborative, evolving journey rather than a mere compilation of code changes*. In essence, this integration transcends the boundaries of quantitative and qualitative methodologies, fostering a research approach that mirrors the complexity of the software development ecosystem. It is an opportunity to see automated analyses and human narratives converging. It offers researchers a holistic understanding of the intricate connection between code, collaboration, and the people who bring software projects to life. In Appendix 6, we report a non-exhaustive list of repositories that the reader may find helpful to running mixed-method MSR research.

A Do for Educators

MSR studies can be combined by leveraging other methods to obtain mixed-method research.

4 Ethical Mining

MSR research typically involves analyzing human-generated data, including developers' activities and interactions in repositories, as well as cultural and geographic data, such as racial and ethnic origin, which are necessary for studies on the geo-cultural dispersion of software communities. Although it is classified as a data-driven strategy rather than a respondent-driven one [54], ethical considerations may still be relevant. For example, consider the case of a mining study aiming to investigate PR acceptance rates in open-source repositories. The study may require ranking contributors based on their PR acceptance rate: while GitHub data may indeed be used to rank contributors and identify the most/least successful contributors, publishing the identity of those contributors would be unethical.

Despite ethics playing a role in MSR research, it is often overlooked. From 100+ papers on MSR mining challenges and data showcases from 2006 to 2021, only a few discussed ethics or data anonymization as part of the threats to validity [23], hence suggesting the need for educating the next generation of researchers to consider ethical aspects while mining software repository data. This is especially true when considering the potential impact of MSR research in practice: according

to Feitelson [21], open-source developers are indeed largely open to research, provided it is done transparently.

Considering source code repositories as an example, it is essential to note that publishing source code under a license differs from publicly releasing a repository. Repository data are often not explicitly licensed for study and unrestricted use. Therefore, ethical concerns might arise, and ethical issues should be considered in such situations [23]. More particularly, when mining the human-generated data coming from repositories, some recommendations based on the Menlo report include [23]:

- *Stakeholder identification*: Consultation of all parties involved and impacted by the research (e.g., ICT researchers, human subjects, non-subjects, users, and platform owners) before using their data for research purposes. Adhering to this guideline when performing large-scale mining analyses may be challenging. Nonetheless, it is worth remarking that, according to the GitHub's acceptable use policy,[14] researchers *"may use public, non-personal information from the Service for research purposes, only if any publications resulting from that research are open access."* This applies to both data directly extracted by GitHub, e.g., through GitHub's APIs, and data indirectly coming from GitHub, e.g., publicly available datasets that derive from GitHub data. As such, one approach is to focus on using non-personal data and ensuring that research findings are published as open access. Additionally, researchers may engage with repository maintainers and community members through public communication channels to inform them about the research and invite feedback. On the one hand, this may help ensure research transparency. On the other hand, this may foster a collaborative environment, even if direct consultation with every individual contributor is impractical.
- *Informed consent*: Ensuring that permission is sought, participation is voluntary, and data is anonymized, carefully processed, stored, and discarded are all essential aspects to be considered. When analyzing large projects with numerous contributors, obtaining informed consent from all participants can be impractical because many contributors might have left or lost interest in the project. In such cases, ethical considerations include respecting the contributors' privacy and ensuring that their data is used responsibly. On the one hand, GitHub's acceptable use policy solely allows the use of public, non-personal information for research purposes. On the other hand, one approach is to anonymize or use pseudonyms for contributors' names, as this protects their identities while still allowing for meaningful research.
- *Risks and benefits balancing*: the need to consider the potential harms, personal data protection, and the impact of the results.
- *Fairness and equity*: the need to consider the fair selection of subjects, data availability, and fair treatment of parties involved in the study.

[14] The GitHub's acceptable use policy: https://docs.github.com/en/site-policy/acceptable-use-policies/github-acceptable-use-policies.

- *Compliance, transparency, and accountability*: legal compliance should be considered as part of data handling. For instance, dealing with personal data may require the researcher to comply with the law of the country they are conducting the research.

Some points to consider when performing MSR studies include:

- A change of mindset from *"Here is a dataset, let us see what we can find"*, as this can be risky for the participants. So, the ethics considerations should assess the potential areas of harm, including the observations and judgments, and evaluate the impact the research results can have on the individuals/developers.
- MSR studies are often conducted on a sample of the data, which is random. Ethics could play a role in ensuring the sample represents the people involved and is inclusive regarding the questions and the process. Ethical consideration ensures that individuals and analyses are not excluded from the results.
- There are different laws and regulations regarding privacy, including the EU's General Data Protection Regulation (GDPR) and the California Consumer Privacy Act of 2018 (CCPA), in addition to the Intellectual Property (IP) laws. Various licenses also state different usage allowances. These laws sometimes restrict specific usages and research and should be considered when designing the study or before data collection. However, for the sake of simplicity, restricting research repositories with licenses that permit studies could affect the generalizability of the research findings. For instance, this aspect should be considered and actively discussed in the paper as a threat to validity.

In conclusion, educators should proactively integrate ethical MSR practices in their courses. First and foremost, it is paramount to highlight the significance of adhering to GitHub's acceptable use policy and safeguarding contributors' identities by refraining from disclosure without explicit consent. Additionally, educators can foster student comprehension of the legal and regulatory landscape governing data privacy and intellectual property rights through interactive discussions and hands-on learning experiences. Employing awareness-targeting teaching methods like quizzes and serious games may be an effective strategy to stimulate critical thinking on ethical challenges inherent in MSR research. An exemplary illustration of this pedagogical approach was provided by Teo et al. [59], who introduced an interactive, scenario-based ethical AI quiz for students to self-assess their awareness and perceptions regarding AI ethics.

Concrete Examples To exemplify how ethical concerns may be effectively addressed in the context of MSR research, we briefly discuss the strategies employed by two relevant articles published at the Mining Software Repositories Conference. The first, authored by Yamashita et al. [67], collected evolutionary data concerned with the programming skills of practitioners to publicly release an open dataset. Before releasing the data, the authors re-wrote the whole change history of the Git repositories so that sensitive information, e.g., developer names and contact details, were removed or changed consistently to protect privacy and confidentiality. The second article, authored by Gonzalez-Barahona et al. [24], released a suite of tools

to extract data from software repositories, also contributing a database of human-generated data coming from open-source communities. In their article, the authors explicitly mentioned that the use of data was allowed by the organization providing those data. These two articles represent two valuable examples of how ethics should be preserved while mining software repositories and may be used by educators as case studies.

A Do for Educators

MSR studies must be compliant to ethical aspects and regulations enforced by policymakers. Educators should proactively introduce ethical MSR practices in their courses, for instance, by employing awareness-targeting teaching strategies.

5 Recent Trends and Future Outlook for Educators Leveraging Mining Software Repositories

As a last part of this book chapter, let us reflect on the recent trends in MSR research, with an outlook on the future development of the field.

The most recent advances made by the research community over the last few years reflect the dynamic nature of software development practices and the increasing integration of advanced technologies. One prominent trend is the growing emphasis on leveraging artificial intelligence (AI) techniques to extract insights from vast repositories of software-related data. For instance, the 2024 Mining Challenge Track of the 21^{th} International Conference on Mining Software Repositories (MSR 2024) has featured a challenge on developers' ChatGPT conversations.[15] Such an example underscores the relevance of AI-driven approaches in understanding developer interactions, collaboration patterns, and decision-making processes. Educators responsible for mining software repository courses can leverage this trend to provide students with hands-on experience in applying AI techniques to analyze and interpret software-related data, fostering a deeper understanding of the complexities of modern software development processes.

Furthermore, there is a notable shift toward exploring the dynamics of software ecosystems within MSR education. This trend reflects the recognition of software systems as complex socio-technical ecosystems comprising diverse stakeholders, technologies, and dependencies. Educators are called to increasingly incorporate modules on software ecosystems into mining software repository courses, enabling students to gain insights into the interconnectedness of software projects, the evolution of software ecosystems over time, and the impact of ecosystem characteristics on software quality and maintainability. By incorporating software ecosystem

[15] MSR 2024 Mining Challenge: https://2024.msrconf.org/track/msr-2024-mining-challenge?

analysis into their curriculum, educators can empower students to navigate the intricacies of real-world software development scenarios and equip them with the skills necessary to contribute meaningfully to software projects within diverse ecosystem contexts.

Finally, MSR education is increasingly emphasizing interdisciplinary collaborations and integrating diverse data sources and methodologies. Educators recognize the value of combining traditional MSR techniques with insights from machine learning, natural language processing, and social network analysis to address complex research questions and emerging challenges in software engineering. By fostering interdisciplinary collaboration and exposing students to various methods and tools, educators can prepare them to tackle real-world software engineering problems effectively and drive innovation in mining software repositories.

6 Sample List of Repositories

- Bugzilla: https://www.bugzilla.org/
- Bitbucket: https://bitbucket.org/
- GitLab: https://gitlab.com/
- Azure Repos: https://azure.microsoft.com/en-us/services/devops/repos/
- Google Code: https://code.google.com/archive/
- Jira: https://www.atlassian.com/software/jira
- ProjectLocker: https://www.projectlocker.com/
- CloudForge: https://cloudforge.com/
- Zenodo: https://zenodo.org/
- Awesome GPT: https://gpt4.tools/
- Docker Hub: https://hub.docker.com/
- Kaggle: https://www.kaggle.com/

Discussion Forums

- Reverse Engineering: https://reverseengineering.stackexchange.com/
- Software Engineering: https://softwareengineering.stackexchange.com/
- Software Quality Assurance and Test: https://sqa.stackexchange.com/
- GenAI: https://genai.stackexchange.com/
- DevOps: https://devops.stackexchange.com/
- Hash Node: https://hashnode.com/
- Dev: https://dev.to/p/editor_guide
- Code Project: https://www.codeproject.com/

Online Coding Platforms Used by Developers

- Code Pen: https://codepen.io/
- Replit: https://replit.com/

References

1. Abdelrazek, A., Eid, Y., Gawish, E., Medhat, W., Hassan, A.: Topic modeling algorithms and applications: a survey. Inform. Syst. **112**, 102131 (2023)
2. Azeem, M.I., Palomba, F., Shi, L., Wang, Q.: Machine learning techniques for code smell detection: a systematic literature review and meta-analysis. Inform. Softw. Technol. **108**, 115–138 (2019)
3. Barros, D., Horita, F., Wiese, I., Silva, K.: A mining software repository extended cookbook: lessons learned from a literature review. In: Proceedings of the XXXV Brazilian Symposium on Software Engineering, pp. 1–10 (2021)
4. Basili, V.R.: Goal, question, metric paradigm. Encyclopedia Softw. Eng. **1**, 528–532 (1994)
5. Binkley, D.: Source code analysis: a road map. In: Future of Software Engineering (FOSE'07), pp. 104–119 (2007)
6. Borges, H., Tulio Valente, M.: What's in a GitHub star? Understanding repository starring practices in a social coding platform. J. Syst. Softw. **146**, 112–129 (2018).
7. Catolino, G., Palomba, F., Zaidman, A., Ferrucci, F.: Not all bugs are the same: understanding, characterizing, and classifying bug types. J. Syst. Softw. **152**, 165–181 (2019)
8. Chatterjee, P., Sharma, T., Ralph, P.: Empirical standards for repository mining. MSR '22, pp. 142–143. Association for Computing Machinery, New York (2022)
9. Chen, T.H., Thomas, S.W., Hassan, A.E.: A survey on the use of topic models when mining software repositories. Empir. Softw. Eng. **21**, 1843–1919 (2016)
10. Chidamber, S.R., Kemerer, C.F.: A metrics suite for object oriented design. IEEE Trans. Softw. Eng. **20**(6), 476–493 (1994)
11. Creswell, J.W.: Mixed-method research: Introduction and application. In: Handbook of Educational Policy, pp. 455–472. Elsevier, Amsterdam (1999)
12. Dalla Palma, S., Di Nucci, D., Palomba, F., Tamburri, D.A.: Within-project defect prediction of infrastructure-as-code using product and process metrics. IEEE Trans. Softw. Eng. **48**(6), 2086–2104 (2021)
13. di Biase, M., Rastogi, A., Bruntink, M., van Deursen, A.: The delta maintainability model: measuring maintainability of fine-grained code changes. In: 2019 IEEE/ACM International Conference on Technical Debt (TechDebt), pp. 113–122. IEEE, Piscataway (2019)
14. de Oliveira Neto, F.G., Torkar, R., Feldt, R., Gren, L., Furia, C.A., Huang, Z.: Evolution of statistical analysis in empirical software engineering research: Current state and steps forward. J. Syst. Softw. **156**, 246–267 (2019)
15. Dey, T., Mousavi, S., Ponce, E., Fry, T., Vasilescu, B., Filippova, A., Mockus, A.: Detecting and characterizing bots that commit code. In: Proceedings of the 17th International Conference on Mining Software Repositories, pp. 209–219 (2020)
16. Dey, T., Vasilescu, B., Mockus, A.: An exploratory study of bot commits. In: Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops, ICSEW'20, pp. 61–65. Association for Computing Machinery, New York (2020)
17. Dit, B., Revelle, M., Gethers, M., Poshyvanyk, D.: Feature location in source code: a taxonomy and survey. J. Softw. Evol. Process **25**(1), 53–95 (2013)
18. Emanuelsson, P., Nilsson, U.: A comparative study of industrial static analysis tools. Electron. Notes Theor. Comput. Sci. **217**, 5–21 (2008)
19. Falessi, D., Huang, J., Narayana, L., Thai, J.F., Turhan, B.: On the need of preserving order of data when validating within-project defect classifiers. Empir. Softw. Eng. **25**, 4805–4830 (2020)
20. Falessi, D., Juristo, N., Wohlin, C., Turhan, B., Münch, J., Jedlitschka, A., Oivo, M.: Empirical software engineering experts on the use of students and professionals in experiments. Empir. Softw. Eng. **23**, 452–489 (2018)
21. Feitelson, D.G.: We do not appreciate being experimented on: developer and researcher views on the ethics of experiments on open-source projects. J. Syst. Softw. **204**, 111774 (2023)

22. Giordano, G., Festa, G., Catolino, G., Palomba, F., Ferrucci, F., Gravino, C.: On the adoption and effects of source code reuse on defect proneness and maintenance effort. Empir. Softw. Eng. **29**(1), 20 (2024)
23. Gold, N.E., Krinke, J.: Ethics in the mining of software repositories. Empir. Softw. Eng. **27**(1), 17 (2022)
24. Gonzalez-Barahona, J.M., Robles, G., Izquierdo-Cortazar, D.: The MetricsGrimoire database collection. In: 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, pp. 478–481. IEEE, Piscataway (2015)
25. Gousios, G., Kalliamvakou, E., Spinellis, D.: Measuring developer contribution from software repository data. In: Proceedings of the 2008 International Working Conference on Mining Software Repositories, pp. 129–132 (2008)
26. Güemes-Peña, D., López-Nozal, C., Marticorena-Sánchez, R., Maudes-Raedo, J.: Emerging topics in mining software repositories: machine learning in software repositories and datasets. Progr. Artif. Intell. **7**, 237–247 (2018)
27. Gupta, M., Sureka, A., Padmanabhuni, S.: Process mining multiple repositories for software defect resolution from control and organizational perspective. In: Proceedings of the 11th Working Conference on Mining Software Repositories, MSR 2014, pp. 122–131. Association for Computing Machinery, New York (2014)
28. Hassan, A.E.: The road ahead for mining software repositories. In: 2008 Frontiers of Software Maintenance, pp. 48–57. IEEE, Piscataway (2008)
29. Herzig, K., Just, S., Zeller, A.: The impact of tangled code changes on defect prediction models. Empir. Softw. Eng. **21**, 303–336 (2016)
30. Herzig, K., Zeller, A.: The impact of tangled code changes. In: 2013 10th Working Conference on Mining Software Repositories (MSR), pp. 121–130. IEEE, Piscataway (2013)
31. Hoda, R.: Socio-technical grounded theory for software engineering. IEEE Trans. Softw. Eng. **48**(10), 3808–3832 (2021)
32. Kagdi, H., Collard, M.L., Maletic, J.I.: A survey and taxonomy of approaches for mining software repositories in the context of software evolution. J. Softw. Maintenance Evol. Res. Practice **19**(2), 77–131 (2007)
33. Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D.M., Damian, D.: The promises and perils of mining GitHub. In: Proceedings of the 11th Working Conference on Mining Software Repositories, pp. 92–101 (2014)
34. Kamei, Y., Shihab, E., Adams, B., Hassan, A.E., Mockus, A., Sinha, A., Ubayashi, N.: A large-scale empirical study of just-in-time quality assurance. IEEE Trans. Softw. Eng. **39**(6), 757–773 (2013)
35. Kitchenham, B., Madeyski, L., Budgen, D., Keung, J., Brereton, P., Charters, S., Gibbs, S., Pohthong, A.: Robust statistical methods for empirical software engineering. Empir. Softw. Eng. **22**, 579–630 (2017)
36. Kovalenko, V., Palomba, F., Bacchelli, A.: Mining file histories: should we consider branches? In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE '18, pp. 202–213. Association for Computing Machinery, New York (2018)
37. Liu, Z., Xia, X., Hassan, A.E., Lo, D., Xing, Z., Wang, X.: Neural-machine-translation-based commit message generation: how far are we? In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE '18, pp. 373–384. Association for Computing Machinery, New York (2018)
38. Mahmood, Z., Bowes, D., Hall, T., Lane, P.C., Petri, J.: Reproducibility and replicability of software defect prediction studies. Inform. Softw. Technol. **99**, 148–163 (2018)
39. Marcus, A., Sergeyev, A., Rajlich, V., Maletic, J.I.: An information retrieval approach to concept location in source code. In: 11th Working Conference on Reverse Engineering, pp. 214–223. IEEE, Piscataway (2004)
40. Mens, T.: An ecosystemic and socio-technical view on software maintenance and evolution. In: 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 1–8. IEEE, Piscataway (2016)

41. Moser, R., Pedrycz, W., Succi, G.: A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In: Proceedings of the 30th international conference on Software Engineering, pp. 181–190 (2008)
42. Munaiah, N., Kroh, S., Cabrey, C., Nagappan, M.: Curating GitHub for engineered software projects. Empir. Softw. Eng. **22**, 3219–3253 (2017)
43. Nguyen, H.A., Nguyen, A.T., Nguyen, T.N.: Filtering noise in mixed-purpose fixing commits to improve defect prediction and localization. In: 2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE), pp. 138–147. IEEE, Piscataway (2013)
44. Nguyen, N., Nadi, S.: An empirical evaluation of GitHub copilot's code suggestions. In: Proceedings of the 19th International Conference on Mining Software Repositories, MSR '22, pp. 1–5. Association for Computing Machinery, New York (2022)
45. Poncin, W., Serebrenik, A., Van Den Brand, M.: Process mining software repositories. In: 2011 15th European Conference on Software Maintenance and Reengineering, pp. 5–14. IEEE, Piscataway (2011)
46. Qiu, H.S., Nolte, A., Brown, A., Serebrenik, A., Vasilescu, B.: Going farther together: the impact of social capital on sustained participation in open source. In: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), pp. 688–699. IEEE, Piscataway (2019)
47. Ram, A., Sawant, A.A., Castelluccio, M., Bacchelli, A.: What makes a code change easier to review: an empirical investigation on code change reviewability. In: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 201–212 (2018)
48. Rao, S., Kak, A.: Retrieval from software libraries for bug localization: a comparative study of generic and composite text models. In: Proceedings of the 8th Working Conference on Mining Software Repositories, pp. 43–52 (2011)
49. Rosa, G., Pascarella, L., Scalabrino, S., Tufano, R., Bavota, G., Lanza, M., Oliveto, R.: Evaluating SZZ implementations through a developer-informed oracle. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), pp. 436–447. IEEE, Piscataway (2021)
50. Salza, P., Palomba, F., Di Nucci, D., D'Uva, C., De Lucia, A., Ferrucci, F.: Do developers update third-party libraries in mobile apps? In: Proceedings of the 26th Conference on Program Comprehension, ICPC '18, pp. 255–265. Association for Computing Machinery, New York (2018)
51. Silva, C.C., Galster, M., Gilson, F.: Topic modeling in software engineering research. Empir. Softw. Eng. **26**(6), 120 (2021)
52. Śliwerski, J., Zimmermann, T., Zeller, A.: When do changes induce fixes? ACM SIGSOFT Softw. Eng. Notes **30**(4), 1–5 (2005)
53. Spadini, D., Aniche, M., Bacchelli, A.: Pydriller: Python framework for mining software repositories. In: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 908–911 (2018)
54. Storey, M.A., Ernst, N.A., Williams, C., Kalliamvakou, E.: The who, what, how of software engineering research: a socio-technical framework. Empir. Softw. Eng. **25**, 4097–4129 (2020)
55. Storey, M.A., Hoda, R., Milani, A.M.P., Baldassarre, M.T.: Guidelines for using mixed and multi methods research in software engineering (2024). arXiv preprint arXiv:2404.06011
56. Storey, M.A., Russo, D., Novielli, N., Kobayashi, T., Wang, D.: A disruptive research playbook for studying disruptive innovations (2024). arXiv preprint arXiv:2402.13329
57. Sullivan, G., Feinn, R.: Using effect size—or why the p value is not enough. J. Grad. Med. Educ. **4**(3), 279–282 2012. https://doi.org/10.4300
58. Tao, Y., Dang, Y., Xie, T., Zhang, D., Kim, S.: How do software engineers understand code changes? An exploratory study in industry. In: Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, pp. 1–11 (2012)

59. Teo, W., Teoh, Z., Arabi, D.A., Aboushadi, M., Lai, K., Ng, Z., Pant, A., Hoda, R., Tantithamthavorn, C., Turhan, B.: What would you do? an ethical ai quiz. In: 2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), pp. 112–116. IEEE, Piscataway (2023)
60. Tufano, M., Palomba, F., Bavota, G., Oliveto, R., Di Penta, M., De Lucia, A., Poshyvanyk, D.: When and why your code starts to smell bad. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, vol. 1, pp. 403–414 (2015)
61. Verdecchia, R., Engström, E., Lago, P., Runeson, P., Song, Q.: Threats to validity in software engineering research: a critical reflection. Inform. Softw. Technol. **164**, 107329 (2023)
62. Wankhade, M., Rao, A.C.S., Kulkarni, C.: A survey on sentiment analysis methods, applications, and challenges. Artif. Intell. Rev. **55**(7), 5731–5780 (2022)
63. Wen, F., Nagy, C., Lanza, M., Bavota, G.: An empirical study of quick remedy commits. In: Proceedings of the 28th International Conference on Program Comprehension, ICPC '20, pp. 60–71. Association for Computing Machinery, New York (2020)
64. Wen, F., Nagy, C., Lanza, M., Bavota, G.: Quick remedy commits and their impact on mining software repositories. Empir. Softw. Eng. **27**, 1–31 (2022)
65. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering. Springer, Berlin (2012)
66. Yamaguchi, F., Rieck, K., et al.: Vulnerability extrapolation: assisted discovery of vulnerabilities using machine learning. In: 5th USENIX Workshop on Offensive Technologies (WOOT 11) (2011)
67. Yamashita, A., Abtahizadeh, S.A., Khomh, F., Guéhéneuc, Y.G.: Software evolution and quality data from controlled, multiple, industrial case studies. In: 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), pp. 507–510. IEEE (2017)
68. Yang, Y., Xia, X., Lo, D., Grundy, J.: A survey on deep learning for software engineering. ACM Comput. Surv. **54**(10s), 1–73 (2022)

Teaching Simulation as a Research Method in Empirical Software Engineering

Breno Bernard Nicolau de França, Dietmar Pfahl, Valdemar Vicente Graciano Neto, and Nauman Bin Ali

Abstract The chapter supports educators and postgraduate students in understanding the role of simulation in software engineering research based on the authors' experience. This way, it includes a background positioning simulation-based studies in software engineering research, the proposition of learning objectives for teaching simulation as a research method, and presents our experience when teaching simulation concepts and practice. For educators, it further provides learning objectives when teaching simulation, considering the current state of the art in software engineering research and the necessary guidance and recommended learning activities to achieve these objectives. For students, it drives the learning path for those interested in learning this method but had no opportunity to engage in an entire course on simulation in the context of empirical research.

1 Introduction

Several fields, such as physics, chemistry, biology, and others, have matured over time and developed knowledge, theories, and laws. Emerging from this evolution, researchers and engineers created mathematical models to investigate certain conditions using computer simulation.

B. B. N. de França (✉)
Breno Bernard Nicolau de França, UNICAMP, Campinas, Brazil
e-mail: bfranca@unicamp.br

D. Pfahl
University of Tartu, Tartu, Estonia
e-mail: dietmar.pfahl@ut.ee

V. V. G. Neto
Universidade Federal de Goiás, Goiânia, Brazil
e-mail: valdemarneto@ufg.br

N. B. Ali
Blekinge Institute of Technology, Karlskrona, Sweden
e-mail: nauman.ali@bth.se

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2024

D. Mendez et al. (eds.), *Handbook on Teaching Empirical Software Engineering*,
https://doi.org/10.1007/978-3-031-71769-7_13

Software engineering (SE), a young field of study compared to classical fields of science, has also evolved and, over time, proposed hundreds of models to simulate software development processes, software systems, and phenomena that can be observed while applying SE in practice. Such a history also yielded and consolidated methodological support for simulation in this field, including processes and guidelines, making it a relevant, contemporary research method in SE [8].

Although recognized as an important research method, we still face the challenge of educating and training young researchers on when and how to apply simulation to investigate SE-related issues. Therefore, in this chapter, we discuss the teaching of simulation as a research method in SE, focusing on simulation introductory concepts and practice. For that, we frame this discussion using a simulation process as background. Moreover, we discuss several aspects related to simulation teaching in the context of SE that we believe to be important for educators and postgraduate students.

In Sect. 2, we start with contextualizing simulation as a research method and defining its role in SE. Sect. 3 sets up learning goals for simulation, considering mainly two contexts: (i) a whole course on simulation and (ii) simulation as one of the topics in a larger course. Section 5 mentions important simulation concepts and discusses how to teach them around the notion of a process for simulation studies. Section 6 discusses issues when teaching the practice of simulation and describes two simulation models used in different contexts during recent years. Section 7 presents challenges and lessons based on the authors' experience when teaching and using simulation as a research method. Additional recommended materials are described and made available in Sects. 8, and 9 concludes this chapter.

2 Simulation as a Research Method in Software Engineering

Simulation, as it is understood in this chapter, is the *imitation* of the behavior of a real-world process or system *over time* [5]. Simulation entails developing a mathematical model mimicking a system or process's real-world behavior. We quantitatively manipulate and evaluate the mathematical model in simulation-based studies instead of changing the real-world system. Figure 1 outlines the role of simulation. Among the different ways to manipulate and investigate a system, the main factors for a trade-off between these alternatives are realism, control, and cost/risk if the introduced change does not produce the expected results or has unforeseen consequences. In this chapter, we only consider digital models, i.e., computer programs that model the structural and behavioral aspects of the real-world phenomenon under investigation. Approaches that use physical models (e.g., scaled-down replicas of a vehicle) to manipulate a system under study for the purpose of exploration or analysis are out of scope. Similarly, mathematical models that do not have a stochastic element, where an analytical solution is more appropriate, are outside the scope of this chapter. Figure 1 highlights the path to a simulation-based study of a system in green color.

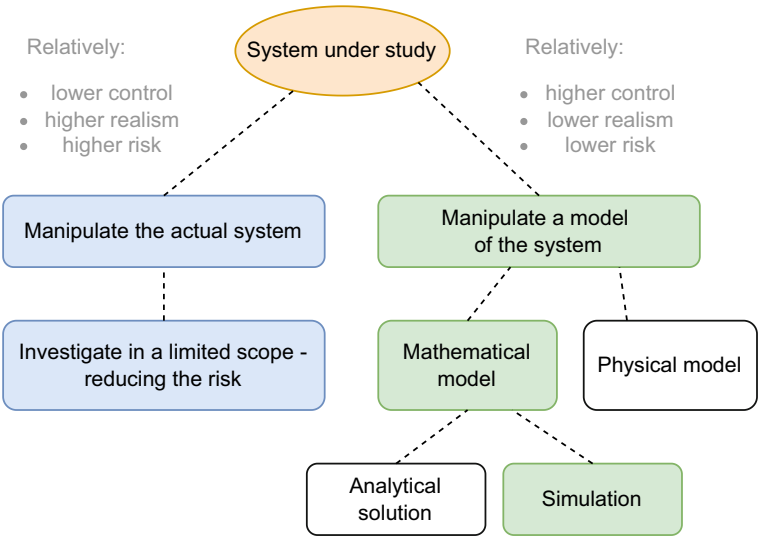


Fig. 1 Role of simulation when studying a system (inspired by Law [19])

When balancing realism, control, and cost/risk, we must weigh the trade-offs of different investigation methods. If human behavior plays an essential role in analyzing real-world processes, no laws of nature are available on which models could be based. Thus, when realism is essential and the associated cost/risk is acceptable, real-world investigations are conducted using case studies and action research. In SE, where knowledge and evidence are strongly context bound, realism is often mandatory, and we often have to compromise on having a lack of control in real-world settings. However, the cost and risk associated with manipulating a product in production or a development process in operation means that we want to be as confident as possible about the likely impact of a proposed change. In such situations, simulation emerges as an economical and proactive tool to forecast outcomes before committing resources to change. Suppose the simulation results do not contradict the expectations based on made assumptions. In that case, there is a higher plausibility that the envisioned changes might also work in usually much more complex real-world settings [26]. To what degree the behavior of the simulation model matches the real-world behavior of the simulated process or product should be stated as a requirement upfront to manage expectations. If a simulation model must closely mimic real-world behavior, thorough validation is essential. This may include expert judgment, empirically derived reference modes to which the simulation output is compared, and other types of checks [19].

Simulation is an invaluable tool for supporting knowledge acquisition and decision-making in research and practice. It enables what-if analyses, even in cases with limited data or a lack of established theories. Simulations can assist by modeling assumptions and approximations, offering objective judgment on

proposed solutions. However, it is essential to use caution to avoid overinterpreting results.

In the context of SE, simulation mostly refers to two different types of systems under study: the software (or system) development process, i.e., *process simulation* [26], or the software product (or system), i.e., *product simulation*. Recently, the term *digital twin* has gained popularity [12]. While digital twins are essentially simulation models, they distinguish themselves by representing an existing physical artifact (product) or real-life process. Moreover, a digital twin is constantly fed with data from the real-life process, often in real time, and used for behavior monitoring and prediction. In this book chapter, we do not restrict ourselves to digital twins but consider the full spectrum of simulation models, including those that are built to analyze and explore the behavior of products and processes that do not yet exist—or at least that deviate from existing ones and are created to assess the effects of implementing planned changes.

Process simulation refers to the simulation of software development processes capturing the flow of work products from requirements to the end product and explaining the use of resources (people and tools) along the process [22]. The goal is usually the exploration of improvement opportunities or the evaluation of the effects of a proposed improvement (i.e., change) of the process organization overall or one or more process elements. More broadly, Kellner et al. identified six categories of purposes of process simulation, i.e., strategic management, planning, control and operational management, process improvement and technology adoption, understanding, and, last but not least, training and learning [17]. The application domains range from requirements engineering to testing and investigating the effect of specific process changes in a single development phase to analyzing multi-project contexts and software maintenance and evolution [35]. A process simulation model would typically capture properties of activities, artifacts, and resources involved in the process. In the context of SE, a process simulation model could, for example, explore the effects of changing the number and/or skills of developers allocated to specific tasks (e.g., coding and testing) on the quality of the outputs of these tasks (e.g., production code and test code) [26].

Product simulation refers to the simulation of a software product or a system, including embedded systems. The goal is usually to evaluate the behavior of the software product or system, for example, in the context of testing. With the rising popularity and fast evolution of AI-based systems, simulation might also play a role in training artificial neural networks (ANNs). Assume, for example, that a cyber-physical system, such as an autonomous vehicle controlled by a trained ANN, is modeled as a digital twin that moves in a simulated virtual world. Then, in the same way, the physical vehicle's ANN might be trained using the data collected by sensors in the real world, and the digital twin's ANN might be trained using the data collected by simulated sensors in the simulated virtual world. If the digital twin and the virtual world closely resemble the physical vehicle and physical world, it should not make a difference whether the ANN is trained using real-world data or data from the simulated virtual world. Both aspects, training of ANNs and simulation-based safety testing, have become hot topics in the context of automated driving systems

(ADS). In recent surveys of approaches for testing ADS [18, 31], simulation-based testing has been identified as an essential method to speed up the testing process by facilitating the exploration of high-risk driving scenarios and the automatic execution of many different test scenarios.

In combination with powerful and often domain-specific simulators, such as CARLA[1] in the specific context of ADS development and testing, both process and product simulation have not only become important in the industry but more and more are seen as research tools, e.g., for exploring the effectiveness of principles of communication and collaboration among software developers, for exploring how design principles may support the maintainability and evolution of software, or for exploring which test scenario selection principles make the testing of ADS most effective and efficient. The potential of simulation as a research tool naturally makes simulation relevant for learning.

3 Setting Up Learning Objectives for Simulation

Teaching simulation as a research method in SE is challenging, as several undergraduate programs overlook simulation and modeling introduction courses. In the ACM Curriculum Guidelines for undergraduate degree programs in computer science, computer simulation is part of a broader discipline called computation sciences [2]. It aims to enable competencies such as formal modeling of real-world situations and use such models in simulations at an introductory level [3]. More advanced courses in this reference curriculum are considered elective, encompassing different purposes of simulation, trade-offs among model attributes such as accuracy and complexity, specific modeling and simulation techniques, verification and validation (V&V), and supporting tools.

This scenario sets a general context for teaching computer simulation as an SE research method in graduate courses. These courses should consider basic and advanced simulation concepts and particular techniques and approaches often used in SE.

As background, the students need previous knowledge of (i) programming and algorithms, as the notions of sequential instructions for reaching a goal is also important in simulation, as well as the notion of abstractions, variables, and control flow, and (ii) introduction to statistics, mainly concepts regarding data distributions and equation modeling; in terms of research methods, (iii) an overview of ESE, experiments, and threats to validity is also important. This last prerequisite differentiates teaching simulation as a technique and an empirical research method. Furthermore, it makes such a course more suitable for postgraduate students. In the case of focusing on the technique perspective, simulation can be (and is referred to by ACM) an undergraduate course.

[1] https://carla.org

In line with the ACM Curriculum Guidelines, we provide the learning objectives for a full course on simulation, considering its use as a research method in empirical software engineering (ESE). As a terminal learning goal, the student should be able *to recognize situations in which simulation is an appropriate research method and apply it accordingly, supported by existing processes and guidelines, when adopting well-established simulation approaches*. Specific or enabling learning objectives include:

1. Explaining the relationship between modeling and simulation, i.e., thinking of simulation as dynamic modeling [Understand]
2. Differentiating among the different types of simulations, including in virtuo and in silico environments [Understand]
3. Explaining the benefits of simulation and modeling in SE research [Understand]
4. Choosing an appropriate modeling approach for a SE problem or situation [Evaluate]
5. Creating a new or extending an existing model of a SE phenomenon and using that model in a simulation [Apply]
6. Verifying and validating the results of a simulation and raising and addressing potential validity threats [Analysis]
7. Performing simulation experiments comparing different scenarios and explaining any differences [Apply]

Another usual scenario is to include simulation as a topic in other courses, e.g., in an ESE course. In such contexts, covering all the topics at their proposed learning levels will be unfeasible. So, the level should be adjusted for each object to accommodate the course expectations. For instance, Course 4 (see Sect. 4) establishes a level of "Understand" with the main simulation-related concepts to understand the whole simulation life cycle (see Sect. 5) and approaches for software engineering (system dynamics and DEVS). Additionally, the two "Evaluate" objectives are useful for (i) decisions when using simulation as a research method and (ii) to assess the quality of this sort of study based on guidelines [10] and the empirical standards [28] on quantitative simulation. This way, the students can review and assess primary studies involving simulation for their theses, but they do not learn how to create models using any particular simulation approach as it would demand several classes for the "Apply" objectives. Regarding the "Apply" level, the students in Course 4 receive training on performing experiments using an existing model (usually a systems dynamics model).

4 Experiences on Teaching Simulation

This section describes the main experiences of teaching simulation as a research method in SE. Sections 5 and 6 mention these experiences from a set of different courses organized by the authors, including **Course 1**, a course on simulation for graduate program students (PhD candidates and master's students) (2019); **Course**

2, a tutorial taught in the European Conference on Software Architecture (2021)[2] [15] and in the Brazilian Congress on Software (2021); **Course 3**, a summer school on the topic (2023); and **Course 4**, an ESE course including simulation as a unit.

Course 1 This 60-hour presential elective course was taught to 13 master's and PhD candidates during the first academic semester in 2019 (March to July 2019). The dynamics were based on (i) expositive classes (16 hours) about DEVS associated with practical programming activities in the laboratory, (ii) student seminars (20 hours), and (iii) practical project development, scientific paper writing, and presentation (24 hours). The learning objectives were to (i) provide subsidies for students to develop studies and projects based on systems simulation; (ii) introduce the fundamentals of modeling dynamic systems through the use of simulations; (iii) present notations, formalisms, and tools used for this purpose; and (iv) enable the students to specify simulation models for representing systems in different domains. The program was based on teaching DEVS with MS4Me, some seminars on chosen articles, and other simulation technologies and paradigms, such as NetLogo for ABS. Besides executing partial practical programming exercises over the first half of the course, during the second half of the course, while they were presenting seminars, they were also required to develop a practical project that should be presented at the end of the course and also communicated in a scientific paper to be submitted to a workshop. Five scientific papers were developed in groups and accepted in the workshop that year.

Course 2 This is a short course of three hours in the format of a tutorial taught twice at the European Conference on Software Architecture (April 2021) and at the Brazilian Congress on Software (September 2021). The audience was students of several levels, professors, researchers, and practitioners who attended those events. Those tutorials were remote due to the COVID-19 pandemic. The content was based on an expositive presentation about the importance of simulation in the German industry by Dr. Pablo Antonino (1 hour) and expositive lessons on DEVS in the last 2 hours. Material and instructions on downloading MS4Me and sample examples were made available to the attendees through a Web site. The learning objectives were [23] to teach (i) knowledge of fundamentals of smart-ecosystem architectures, which was the background of the course, (ii) introduction of DEVS for newcomers using MS4Me, and (iii) introduction of a simulation approach (Dynamic-SoS [21]) and to show how we could automatically generate DEVS code with architectural reconfiguration support and (iv) practical experience of smart-ecosystem simulation, guiding the attendees on how to deploy, run, and interpret the results of the simulation.

Course 3 This was a 1-week remote intensive course of 16 hours during summer in Brazil (February 2023[3]). The audience was students and researchers on many

[2] Slides and supplementary material: https://ww2.inf.ufg.br/insight/tutorialecsa2021/

[3] https://summerschool.inf.ufg.br/en/index.html

levels. More than 40 people from the entire country attended the course. We had an 8-hour course on DEVS with MS4Me, besides a tutorial with guidelines for in silico experiments with simulation (4 hours). In the end, we had a workshop on scientific papers running in parallel with an entire-day hackathon financially sponsored by RTSync[4] to award the best projects and also technically sponsored by TriboCloud,[5] which provided a cloud environment for running the simulator.

Course 4 This is a one-semester ESE course for graduate (PhD and master's) students. It has been offered for 7 years. Its main learning goals are (i) to recognize the established empirical methods, (ii) to properly select research methods, and (iii) how to conduct primary (planning) and secondary (whole process) studies in practice. The course is organized into units that correspond to research methods. One of them is in silico studies, mostly simulation. Before this unit, the students are introduced to ESE, including its origins, paradigms, threats to validity, and other related concepts, controlled experiments, and quantitative (statistical) analysis, which includes measurement scales, descriptive statistics, exploratory data analysis, and statistical inference. This unity includes definitions and motivations for in silico studies, simulation-based studies life cycle and their main activities, particularities of simulation for SE, exemplary studies and models in discrete-event and system dynamics, and guidelines for simulation in SE. Expositive classes, discussions, and seminars are used for activities. The first assessment is the elaboration of a research protocol for a primary study, in which the students define a research goal and should be able to select the appropriate methods to conduct the research. Additionally, one last assessment is the execution of a secondary study. In these assessments, simulation can be used as a research method (primary study), or simulation studies must be assessed in the quality appraisal (secondary studies).

Finally, we are also aware of other initiatives, for instance, teaching ARENA[6] as part of a regular SE course in a Brazilian university [27].

5 Teaching Simulation Concepts

To set up our perspective on simulation, we adopt the following definition from [5]: "Simulation is the *imitation* of the operation of a real-world process or system *over time*. Simulation involves the generation of an *artificial* history of the system, and the observation of that artificial history to draw *inferences* concerning the operating characteristics of the real system that is represented."

Besides the definition, it is important to present and discuss the situations in which simulation suits well for SE research (Sect. 2), as well as presenting

[4] https://rtsync.com/

[5] https://www.tribocloud.com/

[6] https://www.rockwellautomation.com/pt-br/products/software/arena-simulation.html

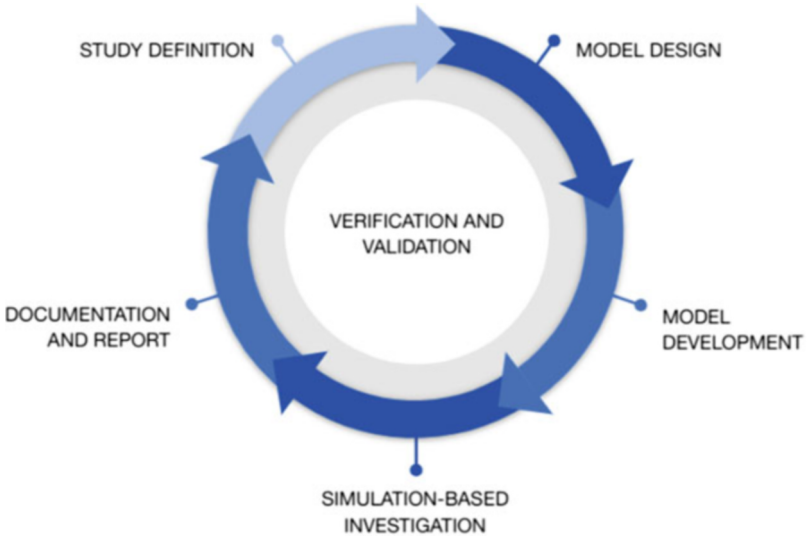


Fig. 2 Simulation-based studies life cycle (Adapted from [8])

motivations for adopting simulation by considering its advantages, such as the low (financial, safety, cost, or even ethical) risk of performing experiments in a highly controlled environment, anticipating "what-if" scenarios, knowledge gaining and theory-building, and possibly working with failure scenarios. Conversely, it is equally important to highlight disadvantages or challenges, such as possible lack of knowledge to develop a simulation model, lack of data for calibration, developing toy or simple models that are not accurate enough for the established research goals, or unfeasibility of validating complex/ambitious models [8].

Simulation-based studies need methodological support, such as guidelines and processes [1], which are important for planning, designing, execution, analysis, and reporting. de França and Ali [8] have discussed the role of such studies in SE research. We present their proposed process, which they further complement with supporting guidelines in Fig 2. We use this process as a framework to structure our discussion of several simulation concepts below.

The first step in the process, the study definition, is similar to other empirical investigations. It includes setting research goals, formulating research questions, and describing the context.

For the next step of model design and development, the students must be exposed to concrete approaches, such as discrete-event simulation (e.g., DEVS) and system dynamics, which are the most common in SE.

Many different simulation approaches exist, with their characteristics, strengths, and weaknesses [14]. The most appropriate approach depends on the simulation goal and the aspects of the system to be simulated, as they capture different

characteristics to be represented or highlighted in the simulation model. Some of the most common approaches (not being restricted to) include:

1. **Discrete-event simulation (DEVS):** DEVS simulation models capture the structure and behavior of a system as a series of events that occur over discrete units of time. It is often used to model complex systems, such as manufacturing plants or transportation systems, where the order of events is important.
2. **Continuous simulation:** This simulation approach replicates the continuous evolution of a system's behavior and finds frequent application in modeling physical systems, particularly those involving fluids and thermodynamics, where the system's state undergoes constant alterations.
3. **System Dynamics (SD):** This is a particular continuous modeling technique focusing on capturing and analyzing the behavior of dynamic systems by incorporating the fundamental principles of feedback and non-linearity. This approach plays a pivotal role in understanding and predicting the dynamics of complex systems characterized by a significant degree of uncertainty. SD finds widespread application in various domains, primarily industrial, management, finance, and social sciences.
4. **Agent-based simulation (ABS):** A simulation approach designed to emulate the intricate dynamics of systems composed of autonomous agents, which can encompass individuals, businesses, or even animals. ABS has found extensive application across various domains, notably economics, social sciences, and engineering.

Considering the associated concepts, the dynamics of these approaches can be taught in traditional expositive classes and readings (see Sect. 8), but the teacher can also use educational games. For example, to learn system dynamics, there is the *beer game*,[7] invented by Jay Forrester, available in multiple formats and is quite simple to apply. Additionally, starting modeling examples and exercises from simpler models and increasing complexity is an important strategy to not overwhelm students with the complexity of large models and non-trivial problems or phenomena to be simulated. Larger and more complex models may be explored once the students have captured the basic concepts of the intended approaches and can be used to work on projects for more practical aspects (see Sect. 6). Finally, including models representing software processes and products is important for the students to realize the range of possible simulated phenomena in SE.

Verification and validation (V&V) of dynamic simulation models include different procedures, most independent of the simulation approach since they take models as a "black box." Theoretically, it is important to differentiate verification from validation [4]. The first is associated with checking implementation consistency and correctness against its specification or conceptual model. Instead, the second implies checking if simulation outcomes resemble real-world scenarios regarding accuracy according to simulation goals.

[7] https://en.wikipedia.org/wiki/Beer_distribution_game

Conceptually, it is important to introduce V&V procedures, how to apply them, and understand when to use each procedure, including their prerequisites in model assumptions, operationalization, and available data. Additionally, each V&V procedure evaluates a model considering particular aspects, and in this sense, they are complementary, and using more than one procedure is desirable. These procedures are crucial in mitigating threats to validity since the simulation model is the main source of threats in this type of study. Not all procedures have established guidelines; some provide only the main idea. This way, presenting examples from the literature on how these procedures were applied to different models is interesting for providing analytical capabilities for the students.

Simulation-based investigations are about performing experiments using valid simulation models. So, it requires some knowledge of experimental design and quantitative analysis (Statistics). If students do not have such background, it is important to provide an overview of experimental terminology and relevant designs (e.g., factorial designs and sensitivity analysis). As experimental designs for simulation can be defined in an *ad hoc* way or based on existing metamodels [10], the students need to understand the consequences of missing factors (or model inputs) or alternatives (possible values for factors), i.e., using incomplete designs. This trade-off can be explored in classes using a simple model and exemplifying candidate designs for experimenting with that model.

Documentation and reporting become simple if the simulation-based study is properly planned. There are guidelines for planning and reporting simulations [10].

Teaching all topics and concepts in each process phase can involve expositive classes, readings, discussions, seminars, and theoretical exercises. One interesting exercise, which can be proposed individually or in groups to foster discussion is to analyze published studies against the process and the guidelines. For operationalizing it, after an expositive class introducing the concepts, the instructor previously selects good and poor examples or simulation studies, assigns them to students, provides the process and guidelines descriptions, and asks for a presentation in seminars where the students should describe the study in an overview along with a critical analysis based on the provided guidelines. This activity allows familiarization with simulation concepts, analysis using a concrete case (study), and critical thinking.

A lesson from giving classes about simulation in SE for years is that exposing students only to concepts in an abstract way, without showing concrete studies, model examples, and potential threats to validity [9], will not unleash their full potential and the critical thinking required for future researchers or experimenters.

6 Teaching Simulation Practice

Teaching simulation practice needs a reference system or behavior so that students can work with modeling and simulation approaches. In this sense, having datasets with reference behaviors or system descriptions may support working with small

projects and exercises. Alternatively, instead of developing simulation models from scratch, students may start from existing models and then evolve these models to answer new questions.

One needs a simulation approach, language, and tools apart from the real system to develop a simulation model. Sect. 5 mentions some simulation approaches, but we remark that several of those may cohabit with the same simulation tool and can be combined to deliver more elaborated results, such as a combination of continuous and discrete simulations (that can be connected through co-simulation or even hybrid simulation models). For instance, MS4Me is a proprietary simulator for DEVS based on Eclipse, but it has a free academic license for students. In MS4Me, we can combine Monte Carlo and DEVS in the same simulation model. Similarly, AnyLogic[8] supports multiple approach models, combining constructs from discrete-event, system dynamics, and agent-based simulation in a composite model. Moreover, we also highlight that some simulation approaches are more appropriate than others when supporting or matching the characteristics of the problem being solved. For instance, SD can be more suitable than DEVS when we intend to model causal relationships due to its inherent feedback loops explicitly described in a model, in contrast to hidden dynamics in event-discrete models, which focus on detailing measurable aspects of entities [29] [30]. On the other hand, if one needs to solve a problem in which there is a composition of entities and the system behaviors are triggered by events, DEVS (as the tool for the solution space) is an abstraction closer to the problem space, reducing the need to a more complex mapping between problem and solution spaces.

The following Sects. 6.1 and 6.2 present two examples of models and how they have been used to support teaching simulation studies.

6.1 A Discrete-Event Model of a Flood Monitoring System

When teaching DEVS analogously to other computational techniques, we need to progress from simple instances until we reach more complex scenarios. DEVS is a formalism based on atomic and coupled models [34]. Atomic models can represent individual entities, such as a software component or an entire system. In contrast, coupled models are combinations of atomic (and other coupled) models, such as an entire system or multiple systems interoperating.

Atomic models comprise the following elements [13]: (i) ports (input and output), (ii) a labeled state machine that executes transitions in response to input and output events (hence governing the system operation), (iii) functions that can be used to process data, (iv) data types, and (v) events. Coupled models are expressed as a System Entity Structure (SES), a formal structure governed by axioms that express how atomic models communicate with each other [34].

[8] https://www.anylogic.com/

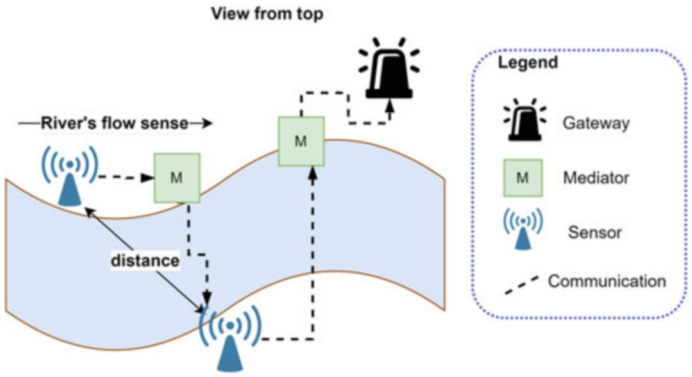


Fig. 3 A flood monitoring system as part of a smart city system [16]

In a prior post-graduation (master's and PhD students) course, the motivational example used to help was a Flood Monitoring System (FMS),[9] a system intended to be part of a smart city. FMS monitors rivers crossing urban areas, which pose a great danger in rainy seasons, potentially damaging property, threatening lives, and spreading disease. FMS notifies possible emergencies to residents, business owners, pedestrians, and drivers located near the flooding area, as well as governmental entities and emergency systems. The FMS we describe concerns a single behavior: *flood alert*. Sensors are spread on the river's edges with a regular distance among them, and mediators exist between every pair of sensors at a pre-established distance between them. The data collected by sensors are collected and transmitted until they reach the gateway. In a flood, the gateway emits an alarm for public authorities. FMS comprises two sensors, two mediators, and one gateway, as in Fig. 3.

Listing 1 illustrates the code in DEVS of a mediator using the DEVS Natural Language dialect. Data types are defined in Lines 1–22 in Listing 1. Duties (which designate gates of mediators) with their respective connections are defined on Lines 8–16. The individual behavior of this mediator is specified in lines 18–23. In short, this behavior defines that after receiving the coordinates of constituents that it mediates (lines 19–20), this mediator will repeatedly receive data from one sensor (line 22) and forward them to the other (line 23). Similarly to gates, duties are transformed into inputs of ports in the atomic model (lines 24–27 in Listing 1). Finally, the behavior is translated into automata in the atomic model (lines 29–42 in Listing 1).

[9] The Flood Monitoring System is a recurring practical scenario that has already been covered in other prior works. In this work, we highlight that we used the scenario as inspiration and modeled the system directly in DEVS. Also, this work differs from previous works that used ASAS [13], which is not a formalism but rather a method for automatically generating DEVS code from SoSADL, that is, a method that involves two formalisms. For the chapter purpose, we reuse the same motivational scenario and describe the corresponding DEVS codes, but not the SoSADL codes.

```
A Distance has a value!
the range of Distance's value is Integer!
use distance with type Distance!

A Abscissa has a value!
the range of Abscissa's value is Integer!
use abscissa with type Abscissa!
A Ordinate has a value!
the range of Ordinate's value is Integer!
use ordinate with type Ordinate!
Coordinate has x and y!
the range of Coordinate's x is Abscissa!
the range of Coordinate's y is Ordinate!
use coordinate with type Coordinate!

A Depth has a value!
the range of Depth's value is Integer!
use depth with type Depth!
Measure has coordinate and depth!
the range of Measure's coordinate is Coordinate!
the range of Measure's depth is Depth!
use measure with type Measure!

accepts input on FromCoordinate with type
     Coordinate!
accepts input on ToCoordinate with type Coordinate
     !
accepts input on FromSensors with type Measure!
generates output on Measure with type Measure!

to start hold in s0 for time 1!
hold in s0 for time 1!
from s0 go to s1! //Unobservable
passivate in s1!
when in s1 and receive Coordinate go to s2!
passivate in s2!
when in s2 and receive Coordinate go to s3!
passivate in s3!
when in s3 and receive Measure go to s4!
hold in s4 for time 1!
after s4 output Measure!
from s4 go to s5!
hold in s5 for time 1!
from s5 go to s3! //Unobservable
```

Listing 1 An atomic model for a Mediator generated in DEVS

In DEVS courses, the first challenge the students usually face is synchronizing the output and input events in both the sender and receiver when modeling two systems or modules exchanging messages. Since both are often represented using atomic models, they are governed by state machines whose state transitions can occur due to input and output events. Since they also occur in specific time frames settled by the modeler, a message exchange between the atomic models is only well succeeded when, in the same instant in time, the sender is sending a message and the receiver is in a state that, when receives some data, triggers the state transition and some processing. Otherwise, it does not work, and the models are not paired accordingly. To facilitate that, we developed a type of pattern for programming in DEVS so that the students could reduce their errors at that phase [24].

Another important difficulty perceived during the first moments with DEVS is related to the specification of labeled state machines. DEVS has a formal

foundation; some students or practitioners are not necessarily used to this type of notation. Then, examples that help with that difficulty are welcome.

When the students overcome this first moment, in general, the newcomers shall understand other details related to the programming resources of the simulation tool, such as receiving multiple messages from different senders at the same moment, and other advanced concepts in DEVS, such as the specification of ports, the manipulation of the content in messages being exchanged between the atomic models, the events processing, selective running, and other features that can be specific to the tool adopted, such as Aspects and Specialization. After these steps, the student is prepared to run a simulation specification process and use the conceived models for experimentation and benchmarking, besides other glimpsed potential applications.

6.2 A System Dynamics Model of Brooks' Law

Another example of a simple and intuitive model to use in classes with a different simulation approach is the model proposed in Chernoguz [7]. It captures the essence of Brooks' law, which states that "*adding manpower to a late software project makes it later*" [6].

Figure 4 depicts the SD diagram of the model, including the *stocks*, i.e., variables in squares that can accumulate values, and *flows*, which are variables looking like a valve linked to stocks so that their values can be increased or decreased based on a

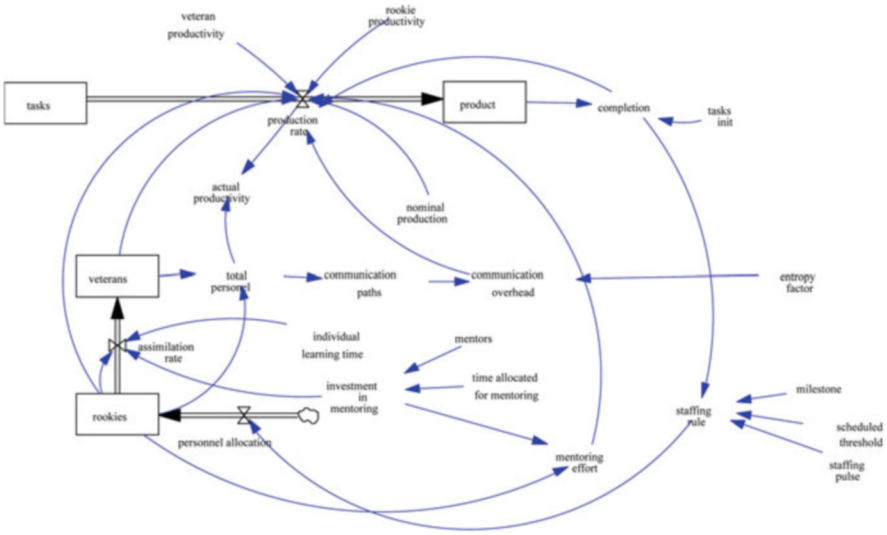


Fig. 4 A system dynamics model of Brooks' law (Based on [7])

rate. The model is implemented using the Vensim PLE tool. Both stocks and flows can be defined in terms of constants or functions. For instance, the stock *rookies* represents the new additional staff allocated using the flow *personnel allocation* rate. This amount of rookies in the stock variable evolves to the stock *veterans* as these gain experience with learning and mentoring through the flow *assimilation rate*. In runtime, the SD engine uses differential equations over those definitions to determine the values at each constant and regular simulation step. All the others are called supporting or auxiliary variables, for example, the variables *individual learning time* and *investment in mentoring*.

This sort of model and diagram is useful for explaining the phenomena under simulation. The causal loops among variables form a chain of reasoning (a form of causal mechanism), so the input changes can be explained via model traces to the output. Of course, the tool abstracts the complexity of the system dynamics execution and formalisms for the students so they can focus on the model variables and their relationships.

In this model, variables that do not depend on other variables are understood as independent. This way, they can be used as input parameters of the model or, experimentally, as model factors. For example, *staffing pulse* is the variable responsible for defining the number of new members to be added at a particular simulation time, based on a *scheduled threshold* that defines the amount of delay to trigger the allocation of additional manpower.

In its original form, this SD model allows the performance of several experiments to understand the effects of allocation strategies, team size, level of team experience, and other variables on development productivity. Additionally, the students may explore model enrichment with additional variables, representing more detailed parts of this phenomenon and even coupling this model to a larger software process/project model.

An example of a simulation-based investigation follows the experimental design in Table 1. In this design, we consider two factors (supporting variables), staffing pulse and entropy factor at four and two levels, respectively. This would lead us to a factorial design with eight runs/trials, considering this is a deterministic model (with no stochastic components). The values for staffing pulse levels were exploratory, considering a baseline (0), and for the entropy factor, we adopted the nominal values from [7].

Table 1 Example of experimental design

Trial	Staffing pulse	Entropy factor
1	0	0.03
2	2	0.03
3	4	0.03
4	6	0.03
5	0	0.06
6	2	0.06
7	4	0.06
8	6	0.06

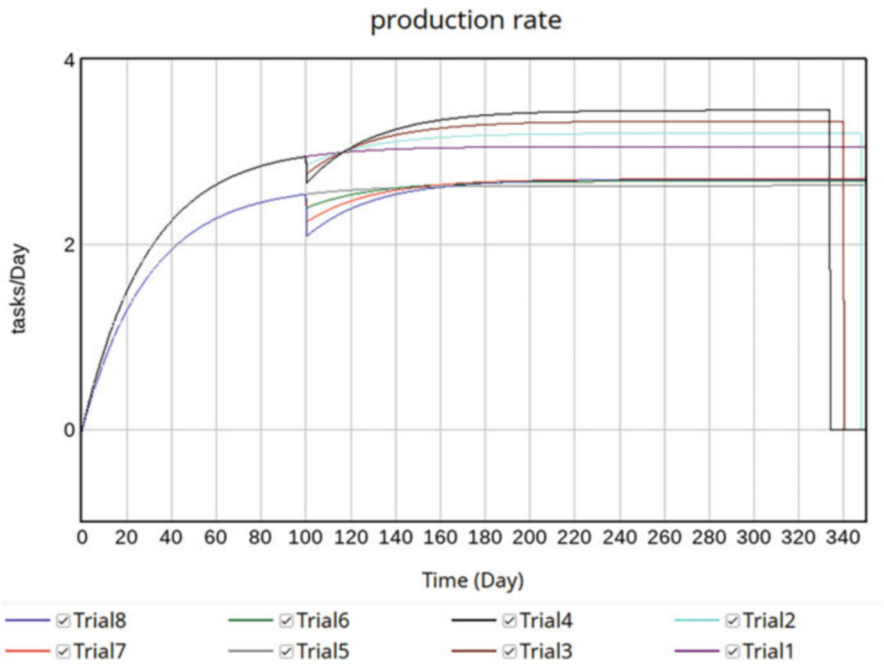


Fig. 5 Output of the production rate variable for eight trials

Defining and evolving an experimental design with a concrete model is also useful for the students to understand the consequences of selecting particular types of designs, such as the complexity of the output analysis and the number of runs. This reinforces concepts about controlled experiments and allows the students to experiment with different designs and realize their advantages and disadvantages in practice.

Figure 5 presents the output of the eight scenarios defined in the design. At all the trials, except for the baselines with no pulse, the behavior is similar: as soon as the project is detected as behind the schedule (at time 100), the staff increases according to the pulse (2, 4, or 6 new members), and it generates an immediate decrease in the production rate. As the rookies gain experience, the production rate increases again until it overcomes the former production rate. This is in line with the statement of Brooks himself when he refers to the law as an "outrageous oversimplification," as the law is applicable in certain conditions, i.e., it is observed mostly in the short term.

The point of this example is not to show a perfect model but to show how students can gain valuable insights from simulating phenomena of interest, gaining knowledge from a "trial-and-error" or exploratory approach, both in the modeling and experimentation perspectives.

7 Challenges and Lessons Learned

Challenges and lessons provided herein come from different experiences from the authors when teaching simulation, including perspectives in which simulation was taught as (i) an empirical/experimental research method or (ii) a general-purpose tool to solve problems, particularly related to critical domains.

Define Your Learning Goals for Simulation Based on Your Course Goals Consider your audience and the course goals when setting up the learning goals for simulation. Our experience ranges from simulation as a topic in an ESE course to a full course on simulation as a general-purpose tool. These settings will differ regarding hours and how in depth the concepts and practice can go for the students or using larger projects, for example.

Run Simulations in Class! Even if your learning goals do not include practical knowledge of modeling or experimenting with models, it is important to show concrete models to students and run simulations during the classes so they can realize the concepts with a simple but solid example. As models selected for teaching are relatively small or simple, educators and students may have no big issues when running these models.

Teach How to Move from the Problem to the Solution Space During an introductory course on simulation, it can be really important to teach how to map concepts from the problem space to the solution space. In other words, abstraction exercises are key for modeling and simulation. For instance, in DEVS, all the problem space concepts must be mapped to atomic models, coupled models, ports, events, and behaviors expressed in a labeled state machine (in the solution space). Hence, several difficulties can emerge, such as mapping the granularities of the problem space concepts to the solution space concepts and converting an intended system behavior to an event-driven state machine or causal loop formalism since many students may lack such knowledge.

Suitability of Simulation Notations for SE Practitioners and Students A survey study with 58 software engineers (in which more than half had not had any contact with simulation models before) revealed that simulation formalism could not be that intuitive for them [20]. The practitioners envisioned several opportunities for improvement, including reinforcements in the visual representation, user support, and model animation, adding support for colors, and the possibility of adding their symbols to represent the explored domain better. The professionals argued that such improvements could reinforce the scalability of the models (visual presentation and execution) and the management of the models and increase the models' reuse for use in fast prototyping practices. Although visual elements are relevant in this case, semantics play an important role in making it suitable or not for a specific audience. The more the notation exposes its mathematical abstractions, the more complex and error-prone it becomes for users. In general, simulation notations were developed for systems engineers. Hence, the notation or user experience often does not conform

to what software engineers use. Transferring this challenge to young researchers (students) in SE is fair. As a consequence, this fact leads to the next lesson learned.

A Process to Incorporate Simulation into Practical Projects While teaching simulation, particularly to software engineers, it is important to teach a basic process that they can follow to incorporate simulation models in their projects [11]. We are aware of SE teams from Brazil and Germany that have adopted the following process [15]: Step 1, Understanding the problem; Step 2, A mapping between problem and space solution; Step 3, A system specification; Step 4, Design of the simulation model by mapping the system specification into the simulation formalism basic concepts; Step 5, Simulation execution, observation, and logging; Step 6, Analysis of results; and Step 7, System development itself. The adoption of such a process was positive for both teams. They use it in industrial project development in partnership with large companies and the Fraunhofer IESE Institute in Germany. In Brazil, it was used to generate successful simulations in projects in the smart cities and space domains.

Highlight the Importance of Having Domain Experts Available for Students As shown in Fig. 2, verification and validation are ongoing throughout the life cycle of a simulation-based study. Validation is often done by reproducing the reference behavior and obtaining feedback from the domain experts. For example, domain expert involvement is necessary for deciding and confirming the scope of the investigation, i.e., deciding which questions the model should be able to answer, operationalizing the model with data from an organization, and documenting and communicating the results. The expert's involvement throughout the life cycle also ensures we achieve a certain buy-in and trust in the model, which is essential for a simulation-based investigation.

The specification of simulation models of the Brazilian space system in DEVS was performed at the Brazilian National Institute for Space Research (INPE). The expert in space systems provided his/her domain expertise, which allows us to model the system precisely [24]. Moreover, the expert also acted as an evaluator for the quality and reliability of the simulation model.

V&V Is Key for Experimentation, But Modeling Is Essential for Gaining Knowledge There is little to no value in trying to provide evidence from a simulation model that has not been validated yet. Knowing the main source of validity threats in a simulation-based study is the model itself; students must be aware that one single V&V procedure is not enough to validate a model. On the other hand, the modeling process in which students will incrementally gain knowledge about the process, system, or phenomenon to be simulated is valuable even initially with a simple simulation model; students can have insights and even discard unfeasible ideas or unrealistic assumptions.

Simulation-Based Investigations Are Experiments and Should Be Systematically Designed Although some simulation outcomes may look reasonable and changing model parameters and running simulations seem simple and easy, students must know that experimenting with models is not about "fishing" the combinations

that look better for a given research question. Exceptional scenarios, factor interactions, and sensitivity analysis should be considered a systematically designed experiment. This way, students need to practice designing experiments based on model parameters.

8 Recommended Readings and Teaching Material

8.1 Readings

- A comprehensive chapter on the role of simulation in software engineering research [8].
- The Modeling and Simulation Body of Knowledge (MSBoK) is also a recent asset and an important reading [25].
- A set of planning and reporting guidelines for simulation-based studies [10].

8.2 Slides

All the slides are available at the Zenodo repository: https://zenodo.org/records/11544898

- Slides of a complete introductory course on DEVS with MS4Me
- Slides of an introductory on modeling and simulation for software architectures
- Slides of an introductory view of in silico studies, including simulation-based ones

8.3 Working Simulation Models

Several simulation models are also available at the Zenodo repository: https://zenodo.org/records/11544898

- Two elementary functional DEVS models (a Hello World) and a reduced version of the Flood Monitoring System as discussed in Sect. 6.1.
- Some DEVS simulation models related to prior publications.
- The simulation model described in Sect. 6.2.
- The Vensim Web site also publishes some sample models, available at https://www.vensim.com/documentation/sample_models.html

8.4 Supporting Tools

- DEVS is one of the most traditional formalisms in M&S [33] and has different free and proprietary implementations. MS4Me is a proprietary Java/Eclipse-based application that RTSync offers, with a free 1-year license and lifelong license (in case you purchase it) and some licenses for academic purposes.
- There is also an implementation in Python [32] and many others that can be consulted.[10]
- Vensim is a simulation environment for system dynamic models. It has the PLE version for educational purposes and other paid versions.
- AnyLogic is a multi-approach, proprietary environment for simulation, including system dynamics, discrete-event, and agent-based simulation.

8.5 Additional Teaching Material

- A reporting standard and evaluation checklist[11] for assessing the quality of a simulation-based study in SE [28].
- Some material is available at https://ww2.inf.ufg.br/insight/tutorialecsa2021/.
- The beer game to learn system dynamics is available at https://beergame.org/the-game/. There are instructions for a board or table game. Also, they have a software version of the game for Windows.

9 Final Remarks

This chapter discusses how teaching simulation could support students, practitioners, and researchers in using it as a research instrument. For that purpose, we compiled a set of important definitions for background, discussed what we understand as the main teaching simulation concepts and principles, and communicated some recommendations based on our prior experience while teaching simulation. We also discussed what we believe to be the main challenges and lessons learned, besides providing supplementary material that teachers can use to teach simulation. We hope newcomers and teachers will consume this material to learn and teach simulation in software engineering.

[10] https://en.wikipedia.org/wiki/List_of_discrete_event_simulation_software

[11] https://github.com/acmsigsoft/EmpiricalStandards/blob/master/docs/standards/QuantitativeSimulation.md

Acknowledgments This work has been supported by ELLIIT, a Strategic Area within IT and Mobile Communications, funded by the Swedish government. The result has also been supported by a research grant for the GIST project (reference number 20220235) by the Knowledge Foundation in Sweden.

References

1. Ali, N.B., Petersen, K.: A consolidated process for software process simulation: State of the art and industry experience. In: 2012 38th Euromicro Conference on Software Engineering and Advanced Applications, pp. 327–336. IEEE, Piscataway (2012)
2. Association for Computing Machinery (ACM) Joint Task Force on Computing Curricula and IEEE Computer Society: Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science. Association for Computing Machinery, New York (2013)
3. Association for Computing Machinery (ACM) Joint Task Force on Computing Curricula and IEEE Computer Society: Computing Curricula 2020 (CC2020): Paradigms for Global Computing Education. Association for Computing Machinery, New York (2020)
4. Babuska, I., Oden, J.T.: Verification and validation in computational engineering and science: basic concepts. Comput. Methods Appl. Mech. Eng. **193**(36–38), 4057–4066 (2004)
5. Banks, J.: Introduction to simulation. In: Proceedings of the 31st conference on Winter simulation: Simulation - a bridge to the future, WSC 1999, Phoenix, AZ, USA, December 05-8, 1999, vol. 1, pp. 7–13 (1999)
6. Brooks Jr., F.P.: The Mythical Man-Month (Anniversary ed.) (1995)
7. Chernoguz, D.G.: The system dynamics of brooks' law in team production. Simulation **87**(11), 947–975 (2011)
8. de França, B.B.N., Ali, N.B.: The role of simulation-based studies in software engineering research. In: Contemporary Empirical Methods in Software Engineering, pp. 263–287 (2020)
9. de França, B.B.N., Travassos, G.H.: Simulation based studies in software engineering: a matter of validity. CLEI Electron. J. **18**(1), 4–1 (2015)
10. de França, B.B.N., Travassos, G.H.: Experimentation with dynamic simulation models in software engineering: planning and reporting guidelines. Empir. Softw. Eng. **21**(3), 1302–1345 (2016)
11. de França, B.B.N., Neto, V.V.G.: Opportunities for simulation in software engineering. In: Anais do III Workshop em Modelagem e Simulação de Sistemas Intensivos em Software, pp. 50–54. SBC, Porto Alegre (2021)
12. Eramo, R., Bordeleau, F., Combemale, B., van Den Brand, M., Wimmer, M., Wortmann, A.: Conceptualizing digital twins. IEEE Softw. **39**(2), 39–46 (2022)
13. Graciano Neto, V.V., Garcés, L., Guessi, M., Paes, C., Manzano, W., Oquendo, F., Nakagawa, E.Y.: ASAS: An approach to support simulation of smart systems. In: 51st HICSS, pp. 5777–5786. Big Island, Hawaii (2018)
14. Graciano Neto, V.V., Kassab, M.: What Every Engineer Should Know About Smart Cities. Taylor & Francis, Lomdon (2023)
15. Graciano Neto, V.V., Manzano, W., Antonino, P.O., Nakagawa, E.Y.: Foundations and research agenda for simulation of smart ecosystems architectures. In: Scandurra, P., Galster, M., Mirandola, R., Weyns, D. (eds.) Software Architecture - 15th European Conference, ECSA 2021 Tracks and Workshops; Växjö, Sweden, September 13–17, 2021, Revised Selected Papers, volume 13365 of Lecture Notes in Computer Science, pp. 333–352. Springer, Berlin (2021)
16. Graciano Neto, V.V., Paes, C.E.B., Oquendo, F., Nakagawa, E.Y.: Supporting simulation of systems-of-systems software architectures by a model-driven derivation of a stimulus generator. In: X Workshop on Distributed Software Development, Software Ecosystems and Systems-of-Systems, pp. 61–70. SBC, Maringa (2016)

17. Kellner, M.I., Madachy, R.J., Raffo, D.M.: Software process simulation modeling: why? what? how? J. Syst. Softw. **46**(2), 91–105 (1999)
18. Khan, F., Falco, M., Anwar, H., Pfahl, D.: Safety testing of automated driving systems: a literature review. IEEE Access **11**, 120049–120072 (2023)
19. Law, A.M.: Simulation Modeling and Analysis. McGraw-Hill Series in Industrial Engineering and Management Science, 5th edn. McGraw-Hill Education, Dubuque (2013)
20. Lebtag, B.G.A., Teixeira, P.G., dos Santos, R.P., Viana, D., Neto, V.V.G.: Strategies to evolve exam notations extracted from a survey with software engineering professionals perspective. J. Softw. Eng. Res. Dev. **10**, 2:1–2:24 (2022)
21. Manzano, W., Graciano Neto, V.V., Nakagawa, E.Y.: Dynamic-sos: an approach for the simulation of systems-of-systems dynamic architectures. Comput. J. **63**(5), 709–731 (2020)
22. Müller, M, Pfahl, D.: Simulation Methods, pp. 117–152. Springer, London (2008)
23. Neto, V.V.G., Manzano, W., Antonino, P.O., Nakagawa, E.Y.: Simulation of software architectures of smart ecosystems: Theory and practice (short paper). In Heinrich, R., Mirandola, R., Weyns, D. (eds.) ECSA 2021 Companion Volume, Virtual (originally: Växjö, Sweden), 13–17 September, 2021, volume 2978 of CEUR Workshop Proceedings. CEUR-WS.org (2021)
24. Neto, V.V.G., Manzano, W., Rohling, A.J., Ferreira, M.G.V., Volpato, T., Nakagawa, E.Y.: Externalizing patterns for simulations in software engineering of systems-of-systems. In: Haddad, H.M., Wainwright, R.L., Chbeir, R. (eds.) Proceedings of the 33rd Annual ACM Symposium on Applied Computing, SAC 2018, Pau, France, April 09–13, 2018, pp. 1687–1694. ACM, New York (2018)
25. Ören, T., Zeigler, B.P., Tolk, A.: Body of Knowledge for Modeling and Simulation: A Handbook by the Society for Modeling and Simulation International. Springer, Berlin (2023)
26. Pfahl, D.: Process Simulation: A Tool for Software Project Managers? pp. 425–446. Springer, Berlin, Heidelberg (2014)
27. Pinheiro, F., Coutinho, E.: A report on the experience of teaching simulation with queuing theory (in Portuguese). In: Proceedings of the 4th Workshop on Modeling and Simulation of Software-Intensive Systems, pp. 21–30. SBC, Uberlândia/MG (2022)
28. Ralph, P., bin Ali, N., Baltes, S., Bianculli, D., Diaz, J., Dittrich, Y., Ernst, N., Felderer, M., Feldt, R., Filieri, A., de França, B.B.N., Furia, C.A., Gay, G., Gold, N., Graziotin, D., He, P., Hoda, R., Juristo, N., Kitchenham, B., Lenarduzzi, V., Martínez, J., Melegati, J., Mendez, D., Menzies, T., Molleri, J., Pfahl, D., Robbes, R., Russo, D., Saarimäki, N., Sarro, F., Taibi, D., Siegmund, J., Spinellis, D., Staron, M., Stol, K., Storey, M.-A., Taibi, D., Tamburri, D., Torchiano, M., Treude, C., Turhan, B., Wang, X., Vegas, S.: Empirical standards for software engineering research (2021)
29. Sweetser, A.: A comparison of system dynamics (SD) and discrete event simulation (DES). In: 17th International Conference of the System Dynamics Society, pp. 20–23 (1999)
30. Tako, A.A., Robinson, S.: Comparing discrete-event simulation and system dynamics: users' perceptions. J. Oper. Res. Soc. **60**, 296–312 (2009)
31. Tang, S., Zhang, Z., Zhang, Y., Zhou, J., Guo, Y., Liu, S., Guo, S., Li, Y.-F., Ma, L., Xue, Y., Liu, Y.: A survey on automated driving system testing: landscapes and trends. ACM Trans. Softw. Eng. Methodol. **32**(5), 1–62 (2023)
32. Van Tendeloo, Y., Vangheluwe, H.: The modular architecture of the python (p) DEVS simulation kernel. In: Proceedings of the 2014 Symposium on Theory of Modeling and Simulation-DEVS, pp. 387–392 (2014)
33. Zeigler, B.P., de França, B.B.N., Graciano Neto, V.V., Hill, R.R., Champagne, L.E., Ören, T.: History of Simulation, pp. 413–434. Springer, Cham (2023)
34. Zeigler, B.P., Sarjoughian, H.S., Duboz, R., Soulie, J.C.: Guide to Modeling and Simulation of Systems of Systems. Springer, Berlin (2012)
35. Zhang, H., Kitchenham, B., Pfahl, D.: Software process simulation modeling: an extended systematic review. In: Münch, J., Yang, Y., Schäfer, W. (eds.) New Modeling Concepts for Today's Software Processes, pp. 309–320. Springer, Berlin, Heidelberg (2010)

Teaching Action Research

Miroslaw Staron

Abstract Action research entered into software engineering as one of the responses to the software engineering research crisis at the end of the last millennium. As one of the challenges in the crisis was the lack of empirical results and the transfer of research results into practices, action research could address these challenges. It is a methodology where collaboration and host organizations are the focus of knowledge discovery, development, and documentation. Although the methodology is often well received in industrial contexts, it is inherently difficult to learn as it requires experience and varies from organization to organization. This chapter describes the pillars of action research as a methodology and how to teach them. The chapter includes examples of teaching action research at the bachelor, master, and PhD levels. In addition to theory, the chapter contains examples from practice.

1 Introduction

The epistemology of action research is rooted in knowledge being a product of collaboration and context. It stems from doing rather than observing or testing. The knowledge developed in action research studies is about improving existing practices by combining these practices with new interventions. When engaging in action research projects, we intend to introduce changes to operations of the host organizations; we want to observe what these interventions (actions) have for consequences, and we want to learn from them.

With this epistemological stance, this research methodology is quite unique as it combines constructive research with observational participatory studies. Action research is defined by Reason and Bradbury as *a participatory, democratic process concerned with developing practical knowing in the pursuit of worthwhile human purposes, grounded in a participatory worldview which we believe is emerging at*

M. Staron (✉)
University of Gothenburg, Gothenburg, Sweden
e-mail: miroslaw.staron@gu.se

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2024

D. Mendez et al. (eds.), *Handbook on Teaching Empirical Software Engineering*,
https://doi.org/10.1007/978-3-031-71769-7_14

this historical moment. It seeks to bring together action and reflection, theory and practice, in participation with others, in the pursuit of practical solutions to issues of pressing concern to people, and more generally the flourishing of individual persons and their communities [15]. It emphasizes the source of knowledge as originating from contemporary problems and the collaboration between participants of the research study.

This is a view that has evolved over the last three decades from the definition of participatory action research [2]: *a form of action research that involves practitioners as both subjects and coresearchers.* It has been adopted in information systems by [4] at the end of the 1990s. It is also the same definition that we use in software engineering [18].

Action research is a methodology that complements experiments, case studies, surveys, and design science research methodologies in software engineering. It has its own merits in its flexibility of the design and learning as well as intervening in practitioners' work. Scientists appreciate the ability to be part of host organizations while conducting their studies, getting the ability to understand the details and peculiarities of the host organizations. Practitioners value the applied nature of the research conducted with their academic partners.

Action research is also flexible, as it allows the action research team to adjust to the situation in the host organization. The flexible nature of action research allows research projects to adjust over time and therefore sustain collaborations over years [17]. As the host organization evolves, the action research teams can change the pursued research questions and learn new things, taking advantage of new opportunities. As the research field evolves, the host organizations can try novel solution to existing problems, taking advantage of the established trust in the action research team and within the organization.

However, the advantages of action research are burdened by certain risks. Since the research studies are embedded in organizations, the action research team needs to understand the generalizability of their findings. The results must be transferable to other organizations and contexts. The transparency of the results must be balanced with their generalizability and the security of the company.

It is because of the flexible nature of the action research that we have inherent problems in understanding it, teaching it to new researchers, and applying it in organizations that operate in highly competitive markets (like AI, telecom, defense, and automotive). First of all, it is difficult to theoretically explain how a good collaboration looks like. Concepts like trust, access to infrastructure, management commitment, or company politics are inherently difficult to explain in theory. One needs to have a firsthand experience to personalize the theoretical knowledge about it. Second, a good action research project is done in cycles, which require time. Long periods are better than short ones as they stimulate building trust and commitment but are also difficult to achieve as reorganizations, promotions, and changing jobs are frequent in our field. Finally, teaching action research is difficult because every organization, collaboration, and country are different in their context, prerequisites, and legal frameworks.

In this chapter, we learn about action research studies, which cross boundaries of academia and industry. Our focus is on collaborations where at least a part of the action team is not employed by the host organization. In practice, this means that in such a setup, the action team is composed of both academic and industrial researchers and practitioners, as opposed to an internal action research project where the researchers and practitioners come from within the organization.

This chapter is structured as follows: First, we explore the pillars of action research, where we focus on the iterative nature of action research and the interaction between organizations and researchers. In Sect. 3, we define each phase of action research and explain what the expected outcomes from these phases are. Since the interaction is done as a team, we then move toward the action research teams themselves in Sect. 4. Then, in Sect. 5, we discuss characteristics of a good host organization and how to interact with it. The teams are responsible for making interventions that are the pivotal element of any action research project. We focus on interventions in Sect. 6, where we discuss what is a good intervention and what is not. We also dive into the question of ethics of action research in Sect. 7. Finally, we conclude the chapter with a set of guidelines on how to teach action research in Sect. 8.

Let's dive into the topic and define the pillars of action research—what makes it so specific? But before that, let us start with the first teaching guideline.

Teaching Guideline

Action research's practical and industrial nature means that we need to teach it in a specific way. I recommend using the case-based approach advocated by Harvard Business School.

To introduce the action research, I recommend reading and discussing one of the existing articles where action research has been used, e.g., [7]. The following aspects should be discussed:

- What is the goal of the study?
- What is intervention/action in the study?
- How is the collaboration between the researchers and the host company organized?
- Which data collection methods are used in this article?

The discussion should focus on the connection between research and practice; it should be about how the knowledge is produced and where state-of-the-art research meets industrial practices.

2 Pillars of Action Research

Action research is centered around the **interventions or actions**[1] (see Fig. 1). The interventions are the part where we do something in order to learn from it. Without the intervention, there is no knowledge production, but the interventions are not the only element that is important for the action research project—we need to prepare for our intervention.

Without actions/interventions, there is no action research as we do not make any changes or adjustments to how the organizations work. In action research, knowledge is derived from observing the effects of these actions/interventions.

Actions/interventions must be done *in situ*, and therefore the **host organizations** are equally important. The host organizations must be prepared for entering the action research projects, and they must understand the consequences of it, the consequences being that there are inherent risks in changing processes, methods, tools, or operations of the host organization. Without them, there is no action research, because we must conduct our actions somewhere.

Finally, the **action team** is a special kind of team that includes both researchers and practitioners with distinct roles. They must be part of one team, and they must see this team as "their" team in the sense that both the researchers and the practitioners are inside the team.

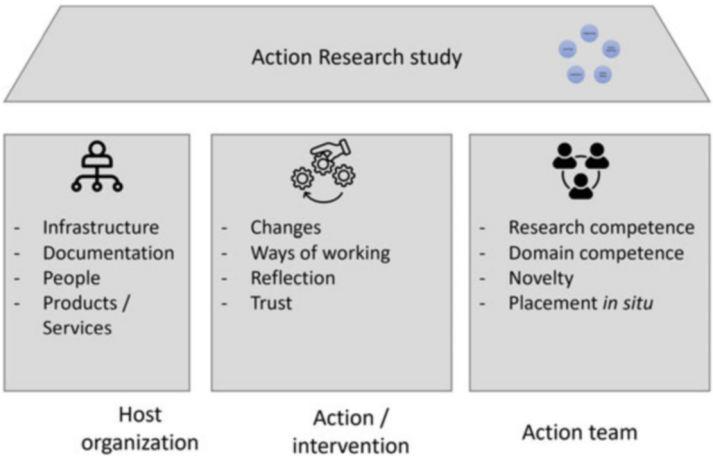


Fig. 1 Pillars of action research: embedding in the host organization, conducting actions/interventions, and collaborative, cross-organization action teams

[1] In this chapter, I use the term action and intervention interchangeably. Although action research has *action* in the name, I often prefer the term *intervention* to emphasize that the action must have an effect on the host organization. I do this to avoid confusion between actions in action research with other types of actions that do not necessarily are interventions.

The researchers must be an internal part of the team in order to be able to assist in conducting interventions at the host organization. If not possible, they can take peripheral role when taking the intervention (e.g., when it is not possible for the researcher to directly participate in such activities as programming, designing, etc.). They must understand and experience the host organization with its decision structures (formal and informal), processes, products, and customers. Only then the researchers can understand which actions can be ethically conducted and which should be ceased.

The practitioners must also see the action team as part of their organization as they must conduct the interventions. Planning, conducting, and evaluating interventions must be included in the work of the practitioners in order to be effective.

Important

Action research is not the only research methodology that focuses on collaboration with industry and introduction new tools. Design science research [25] is another one.

Although these two methodologies share certain characteristics, their epistemological stance on the knowledge creation is very different.

In action research, the focus is on the three pillars in Fig. 1: intervention, action team, and the host organization. In design science research, the focus is on the artefact that is being designed, its function, and usability.

When teaching action research, it is important to contrast these two methodologies and explain that the knowledge produced in the action research projects does not have to be in the form of new designs or artefacts but can be about processes, organizations, tools, and many more. When teaching action research, I provide my students with the following rule of thumb:

If you can pivot on the solution, you are doing action research; if you can pivot on the organization, then you are using design science research.

In practice, this means that in action research, we are not bound to a specific tool that needs to be used or evaluated. We can change the focus on an action research cycle and focus on the ways of working rather than tools used at the host organization. However, if we must focus on the artifact, then we cannot pivot (change) the focus, but we can change the organization, which we collaborate with in order to evaluate our tool, which makes it a design science research cycle.

2.1 Knowledge Produced During Action Research

In action research, the knowledge that is produced during the project can be of three different types (mainly):

1. Process knowledge – development of new methods for developing, testing, and designing software and ways of working at the host organization [6, 7]
2. Organizational knowledge – development of new ways of organizing work at the host organization [9]
3. Construction/tooling knowledge – development of new ways of designing or evaluating software using tools [5, 14]
4. Artefact/models knowledge – development of new artefacts or models that improve the organization's ways of working

In action research studies, the introduction of new methods and tools provides two different opportunities. The first one is the ability of the researchers to test new ideas in practice. The second is the ability to have custom-made methods introduced to the industrial practice.

The latter—custom-made methods—allows us to bridge the gap between academic software engineering and industrial practice [8].

Teaching Guideline
When teaching the epistemology of knowledge in software engineering, I find the best tactics to be by example. We take different types of knowledge that is produced from literature. For example, we take the examples of creating methods from Dittrich et al. [7], while we take examples of creating products from Staron et al. [14].

The most important for the students is to understand that we can create different types of knowledge in action research. We must explain that action research is flexible but that knowledge production must be planned in advance; otherwise, it is not systematic and can be prone to problems with quality, replicability, or reliability.

3 Phases of Action Research

Action research is a cyclic methodology with several phases in each cycle. The number of phases can differ, depending on different schools of action research. In software engineering, the most common is the five phases cycle, depicted in Fig. 2.

In the diagnosing phase, we start with addressing the following question: *What is the real problem that we should address in this cycle?* Although the question is

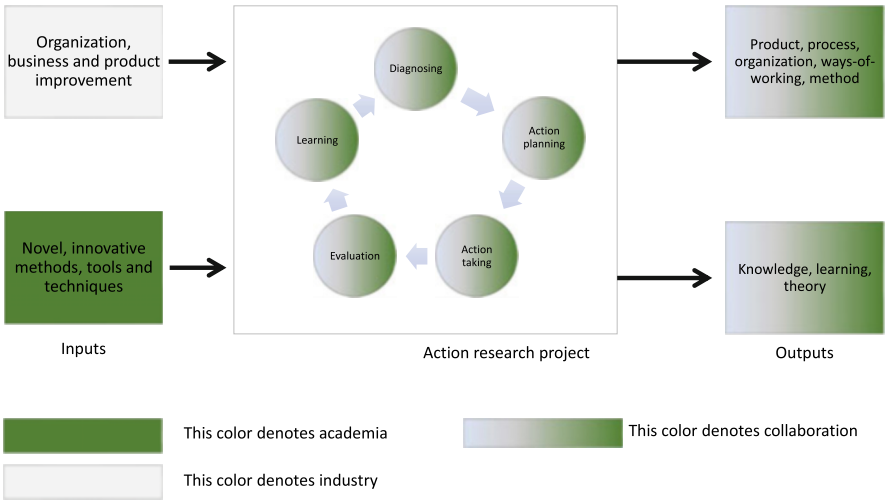


Fig. 2 Phases of action research

often partially answered when initiating the project (or from the previous cycle), it's important to specify which part of the problem should be addressed in each cycle.

The first phase of each action research cycle—diagnosing—is unique for action research. Instead of starting a project with a detailed problem formulation, action research recognizes the fact that one needs to be embedded in the context in order to elicit the problem correctly. Therefore, every action research cycle starts with a precise diagnosis of which problem should be solved. Action researchers should start by collecting opinions and symptoms they need to explore in order to decide which challenge to address during the action research cycle. It's important that the researchers focus on discussions with the practitioners when exploring the context and deciding what to do. The problem to be solved in each cycle should be limited in scope, and its effects should be measurable (or at least observable).

Melin and Axelsson [12] recognize two types of identifying research problems:

1. When an action researcher identifies the problems, i.e., research-driven initiation
2. The problems are presented to the action researcher, i.e., problem-driven initiation

From my experience, the first type, i.e., research-driven initiation, is more common for the diagnosing part, whereas the problem-driven initiation is more common for the overall definition of the research project. Avison et al. [3] recognize the possibility of both parties working together in recognizing the research problem, i.e., collaborative research initiation.

Example
In my work, one of the first action research projects was initiated by the company—problem-driven initiation [22]. The company wanted to improve predictability of the release dates of their product. I was involved in another project, where my role was to design a measurement program.

Since the problem was relevant for my project and it was relevant for the company, we started working on it. We called it the *Release Readiness project*. The diagnosing phase led us, later, to a more detailed definition of the problem and solving it.

In the action planning phase, actions/interventions are planned in a single cycle is always done in a collaborative manner. Academic researchers, industrial researchers, and practitioners need to work together to decide who does the actions and when.

The collaborative nature of the action planning phase provides a unique opportunity for both practitioners and researchers to engage in discussions. The discussions are often aimed at finding ways to solve the problem diagnosed in the first phase and identify resources, products, and processes to be investigated and adjusted.

In the action planning activity, the action team (which is how I call the research team) discusses their plans with the reference groups and needs to get approval for the required resources from the management team. The plans need to be aligned with theoretical foundations of the work, i.e., the action team needs to identify theoretical or empirical work relevant for the diagnosed problem and plan the actions accordingly.

In this phase, the action team, together with the reference team, makes the plans for which data should be collected, from which objects, using which tools. The team also plans which analysis methods should be employed to assess whether their actions lead to solving the diagnosed problem.

Often, although far from always, the action team plans their actions using standard project planning tools, like Gantt charts and work breakdown structures. However, these are often lightweight and documented only internally for the action team to follow and use as a communication tool to management.

Example
In my release readiness project, action planning was done in a form of a workshop. Together with the stakeholder (manager responsible for the release), the action team, and the reference team, we brainstormed which metrics should be used for our predictions. During this workshop, we identified and discussed various relationships between the metrics in order to capture the relevant empirical phenomena that impact the release—e.g., test progress or current quality of the product.

In the action taking phase, the team conducts the interventions—they change the processes, tools, or methods of the host company in order to systematically collect data.

The phase is executed according to the plans laid out in the previous phase and is conducted by the action team. The reference group is involved on a regular basis to provide feedback and to help the action team solve the challenges that they encounter [1].

The action taking phase is specific for action research as it is one of the research methodologies where making changes are allowed, midst in the operations. It's called a flexible research design methodology [16]. For example, the action team is allowed to change the ways of working for software development teams and observe these changes. It is important to note that the action taking phase is both about making the change and observing its effect. As action research is a quantitative methodology, data collection activities provide the possibility to reduce the bias of subjective observations and provide quantitative evidence. This quantitative evidence is used in the next phase—action evaluation—to assess the results of the action/intervention.

Example
This phase of our release readiness project gravitated around collection of data, calculation of metrics/indicators, presentation to the management, and adjusting resources in the project. We collected data weekly and presented them to the stakeholder. The stakeholder made the decisions about the potential re-allocation of resources to fulfill the organization's goals.

In the evaluation phase, the action team analyzes the data collected from the previous phase. The team uses statistical methods to make the analyses and presents the results to the reference team and the management.

In case the data shows that the diagnosed problem is indeed solved using the actions taken, the outcome is straightforward. If the data is inconclusive, the action team either needs to plan for additional analyses and additional data to be collected or needs to pivot, i.e., finalize the current cycle, specify learning, and find a new diagnosis of the problem given the new data collected. Then the action team continues with the next cycle to address this diagnosed problem.

In the evaluation phase, the action team could either take the quantitative approach—use the same statistical methods as experimentation, i.e., descriptive and inferential statistics—or the qualitative approach, use grounded theory, thematic analysis, or workshops. The action team also needs to assure that the analysis of their data is aligned with the theories used in the cycle. This is important in order to make the contribution to the theory building in the next phase.

Example
To evaluate the release readiness indicator, which is how we called the metric developed in the project, we collected statistics of decisions made during the project. We kept track of all changes that the stakeholder took and how they impacted the organization, the product, and also the release readiness indicator (and the metrics used to calculate it). We conducted a workshop in the same group as for the action planning and complemented the quantitative metrics with the qualitative assessment of the results of the project.

In the specifying learning phase, the documentation and generalization of learning takes place. It is done both as practical guidelines for the involved organizations and contexts and as theory building for the research community.

The practical guidelines are often specified in terms of guidebooks, white papers, and instructions at the company's Web. For example, software development teams often use wikis to specify good practices and document good examples. That's often when the results of action research cycles can be found.

The contribution to the theory building is often specified as scientific papers, with the scientific rigor and relevance. It is often the case that these are documented as experience reports from industrial studies [14].

Example
In the case of our release readiness project, we presented the results to a larger audience on an annual conference of the entire company. We also published a paper and presented it on an international conference [22]. We prepared internal information for other stakeholders who adopted this method for other parts of the organization and other products.

4 The Research Team: The Action Team

Action research can be conducted by researchers within the host organization or the natural environment of the participants. In fact, this is how the concept of action research appeared [10]. However, this is not how action research is conducted in software engineering. In software engineering, we usually combine academic researchers with industry professionals. In this context, we can refer to this group as the *action team*—which is the research team that includes (academic) researchers and (company) practitioners. I deliberately use brackets for academic and company, as the boundaries can be fuzzy. For example, an academic researcher may be

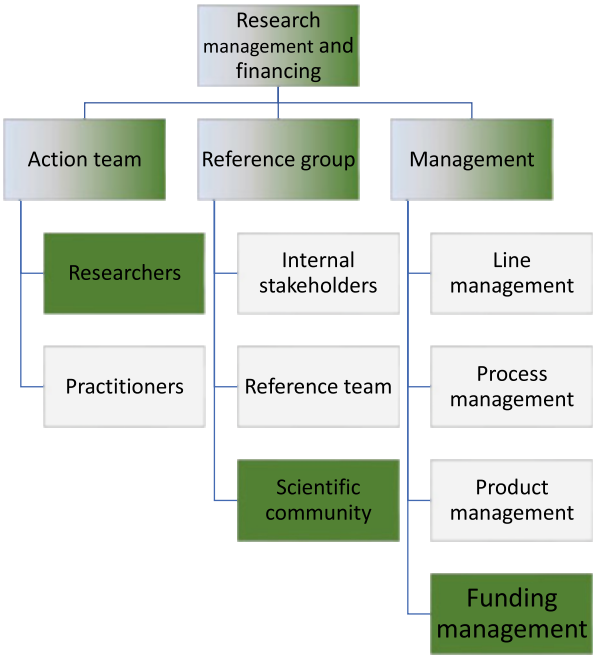


Fig. 3 Action team in the context of other teams. The action team is the one conducting the research. The reference team is the team who helps in evaluating and scaling up the work of the action team. The management team holds the responsibility for financing, staffing, and monitoring the action research study

employed by the company for the duration of the project in order to handle the IPR (Intellectual Property Rights) in a simple way.

Here, we focus on the typical software engineering context—where the researchers and practitioners collaborate in a joint project. In this case, the action team consists of two roles and is surrounded by other stakeholders—the reference team and the management team, as presented in Fig. 3.

In the **action team**, researchers and practitioners have different roles, despite the fact that they work in the same team.

The **practitioners' primary task** is to work with the interventions. Since the practitioners are working at the host organization, they can conduct the interventions/actions. They can change their ways of working and provide data through these changes. They can (and should) participate in the action planning, evaluation, and learning too, but their expertise is crucial for the interventions. The practitioners bring in the following to the action team:

- Internal competence about the company's processes, ways of working, and decision structures
- Possibility to use tools and information from inside of the company, which can require employee-only access
- Historical knowledge about the company's processes, products, and strategies
- Possibility to internally champion the action research study and in particular the action/intervention

- Possibility to invite others to join the relevant parts of the action research cycle, e.g., bring in additional software developers during the evaluation of the outcome of the intervention

The **primary task for the researchers** is to prepare plans for the actions/interventions, observe how they are conducted, collect the data, and analyze it. They can (and should) participate in the interventions, but since they are often not part of the host organization (at least not to the same extent as the practitioners), they cannot conduct the interventions alone. The researchers bring in the following to the action team:

- External (to the host organization) competence about the newest research results, tools, and technologies
- Possibility to test tools outside of the organization before the action team decides (during action planning)
- Systematic planning of data collection—during baseline (action planning, before action taking), during the intervention (during action taking), and after the action taking
- Objective methods for analysis of the data collected during the action research cycle
- Possibility to replicate the study at other organizations (or even open-source products/teams) to study the generalizability of the results obtained at the host organization

The role of the **reference group** is to provide the possibility to get feedback on the progress of the project and to reduce biases. The reference group also helps the action team diagnose the problems and therefore steers the project in the right direction. As the action team is conducting the research, they are biased toward a positive outcome of the project. The reference team is responsible to provide the action team with the feedback on how to reduce this bias and identify when the bias is jeopardizing the outcome of the study.

Finally, the **management** of the company is important as they decide upon the resources needed for the project. The resources, in turn, determine the scope of the project. The product and process management are important as they help support the project in making the right impact of the results of their actions.

Teaching Guideline

Teaching team dynamics is beyond the scope of this chapter, but it is crucial for long-term collaboration.

When teaching collaboration, it is important to consider both academic and industrial perspectives. From the academic perspective, we must discuss the quality of research, including publication venues, targets, and academic goals.

(continued)

This involves understanding how to produce high-quality research that meets academic standards and contributes to the body of knowledge in the field.

From the industrial perspective, it is essential to discuss the practical implications of implementing technology. This includes understanding how to scale a studied technology from a small-scale application within the host team to broader use in other teams and organizations. It is important to differentiate between research prototypes and fully developed software engineering tools and methods. This discussion should cover the challenges and strategies for transitioning from experimental research environments to real-world applications, ensuring that technologies are robust, scalable, and ready for industry adoption.

5 Host Organization

Selecting the right host organization has a crucial effect on the success or failure of action research studies. Although there is no universal template for a suitable organization, there are several characteristics that are important for successful action research collaboration.

The first and foremost condition for the success of the action research project is **the organization's willingness and ability to implement interventions**. Even the best research result is not very useful if it is not implemented at a company, which is the core of action research studies. In the context where the collaboration happens across borders of two entities—university and company—we can achieve this in a few ways:

- Researchers spend a significant time at the company. They are part of the host company operations regularly spending 2 to 5 days a week at the company. This creates the possibility to have a real team spirit and an atmosphere where questions can be asked (and answered) directly and immediately.
- The practitioners who are part of the action team communicate daily with the researchers (and vice versa). In post-pandemic times, online collaboration tools can be a great way of quick communication. Daily, quick, and agile check-in meetings help both move the project forward and create an atmosphere of transparency, trust, and being one team.
- The action team has weekly meetings where the project is discussed and followed up. Stand-up meetings are a bit orthogonal to the previous two, but they are the least that should be done in the project to ensure that things are moving forward.

Naturally, the best option is the first one, where the researcher can participate in the operations of the company. Let me provide an example why this was important in my career.

Example
When I do action research studies, I aim for the first option—being part of the host organization's operation. Let me bring up one experience where I worked with one company on developing measurement systems [20, 24]. In this collaboration, the host company asked me to spend 50% of my time on their premises. Although it sounded strange at the beginning (and foreign to my academic colleagues), this turned out to be a great way of co-creation of research and innovation.

During the time of my stay, I could immediately ask my industrial colleagues about the practices we focused on in our research, and they could learn what kind of research questions and methods are important. One example of my question was, *What's the difference about a defect found in a system test vs. a defect found in a unit test?* An example of my colleague's question was, *Why do we need to formulate our research question "To which degree can we predict defects?" and not "Can we predict defects?"*

Although the above questions seem trivial, we needed to ask them to learn. Only sitting together for a significant part of our work enabled (and encouraged) us to ask these kinds of questions to one another. In the long run, asking questions leads to increased trust and to more fruitful collaboration.

The organization provides the research team with access to the information needed to conduct the action research study. The action team must be able to collect data about their interventions from source systems, and therefore, access to the information should be unrestricted. In practice, this means that everyone in the action team should be able to have the same access for the information relevant to the project. There are a few ways to enable that:

- The researcher uses company's infrastructure for the project in terms of the computer equipment and IT accounts. The researcher must sign the relevant non-disclosure agreements and other legal documents. The same is true for her/his university. The information that is stored at the company's infrastructure is legally under the company's jurisdiction, and therefore the researcher does not have to consider additional security. Analyses, notes, and pre-published materials can be safely stored at the company until they are assessed to be made public. This option allows the action team to work unrestricted until they need to prepare a publication, then the relevant management at the company can provide the team with the feedback on what can (and cannot) be published. It is also easier for the researcher to understand what is possible to published based on the daily work, which often leads to a smooth approval process at the company side.

- The host company agrees upfront which information is relevant for the project and shares that information. If information cannot be publicly shared, then practitioners (who are part of the action team) must actively analyze it, discuss it with the team, and then get it approved. This can be achieved when different participants of the action team have different roles in the team—the practitioners must take more responsibility for the research part. In practice, this means that the analyses are done on company computers, but the researchers can provide an open version of the toolkit, for example, by receiving mock-up data in the right format.

The ways of accessing the information is not as important as the information and collaboration itself.

Example
In my work, I usually try to ask for the company's infrastructure for my action research projects. When being part of the company's team (as I mentioned in the previous box), this is easier both for the company and for me. I'm ensured to be able to use the information when I prepare for actions (diagnosing and action planning) as well as after the action (evaluation and learning).

My colleagues do not have to constantly consider the information's security class, and when we prepare our publications, we can keep the managers informed and "in the loop" so that we know which information can be sensitive. I also prepare a so-called open replication package where we create a mock-up data (or collect it from a relevant open-source repository at GitHub) in order for others to use our research.

The organization has **trust in the action team and is therefore transparent** in the collaboration. Trusting someone is naturally harder than it sounds. It is a combination of both the skills and the so-called personal chemistry. Not everybody can be a good action researcher, and not everybody can be a good action research practitioner. However, there are a few good practices that help develop trust in the team.

- The action team is the researcher's priority. They should consult the team before seeking academic advice. They should also communicate frequently and openly with the team, even about minor issues. Asking questions shows curiosity and builds trust, even if the question seems simple and silly.
- The team should honor their commitments. They should follow through with the agreed activities and report on their outcomes or deviations from the planned outcomes. Sometimes, trying is worth as much as succeeding, as we are in the context of research and knowledge production. Sometimes, learning that something does not work is even more important than forcing a success.

- The team should respect the reality of the host organization. They should acknowledge the diversity and complexity of the people, products, and processes involved in the action research project.
- The team should learn from their mistakes. They should not hide or ignore errors but rather discuss them openly and constructively. Transparency and accountability are key to effective work in the action team.
- The team should treat everyone with respect. They should recognize that working with companies means working with human beings, not abstract entities. They should appreciate the goals, schedules, and abilities of the people in the organization, and not impose unrealistic expectations or demands.

Example
Learning from mistakes is crucial. How we admit and fix them matters most. As a senior researcher, I still make errors sometimes. For instance, I once left an index column in the analyses, which gave great results. But when I checked them, I realized my mistake and told my colleagues. It was hard because they trusted my skills and the results. But when I confessed and showed the worse results, we became more trusting, published the paper, and still collaborate.

We must know and respect the people in the organizations, as they have their own goals, schedules, and abilities. Partaking in an action research study means interest and willingness to collaborate, not a legal contract to deliver.

Teaching Guideline
When teaching the aspects related to the host organization, we need to focus on selecting the host organization. In particular, discussing the criteria for when a collaboration can be profitable for both the academia and industry. In order to do that, we can study guidelines for industry-research collaborations [19, 26].

We also need to teach how to conduct daily communications and meetings. For example, we can simulate this by conducting these meetings during every course meeting. We can ask each of the participants to report on their activities, related to the course, during the past week.

We must discuss the integration of the research activities into the operations. In particular, we should explain that action research must contribute to the industrial practice; we should also discuss methods for identifying problems to study at the host organization.

6 Interventions

Although the research team and the host organization are extremely important, it is the intervention (also called action) that is in the core of action research projects. The intervention is important as it is what gives origin to the knowledge that we generate through evaluation and learning later on. We need the interventions in order to understand how the organization reacts to the changes that are introduced in the project, as well as to collect data to analyze. This could indicate that an intervention could be any kind of activity involving the host organization, but it's not that simple.

A good *intervention is when we introduce a change in the host organization in a systematic and planned manner that has impact on its operations*. The key elements of this definition are:

- **Planned** intervention, which means that the action team must prepare the action and the host organization must be prepared for that action. We must also choose the right timing for the intervention.
- **Systematic** intervention, which means that we need to establish infrastructure for collecting data from the intervention. The action team must collect data before, during, and after the intervention. Before the intervention, the team must establish a baseline, which is used for comparison with the data collected after the intervention. During the intervention, the action team must collect the data about the impact of the intervention on the operations.
- **Impactful** intervention, which means that it has observable effects on the host organization. The impact can be in terms of change or the ways of working, introduction of a new tool, or using new type of data in decision-making processes, for example. The key is that it is part of the processes in the organization, not alongside.

Although it becomes clear what a good intervention is over time, we can provide a few good and bad examples of interventions.

Good examples:

1. Using a new tool to the company operations. When we introduce the tool, we make change in the host organization's processes. We can establish a baseline before the introduction of the tool, observe the introduction itself, and observe the effects of the introduced tool afterward.
2. Using new information in a process of making a decision. When we use the new information, we change the process of making the decision. We can establish a baseline before, observe how the new data is used in the process, and observe the effects of the decision afterward.
3. Changing the process of handling tickets—migration from e-mails to JIRA tickets. As it is a clear change, we can establish a baseline before, observe the change, and then observe the effects afterward.

Example
One example of the intervention from one of my studies is using analysis of the source code to improve coding guidelines [14]. The analysis of the source code was the preparation for action taking, and the actual action taking was the use of the results in the organization. The effect was the improvement of the coding guidelines at the company.

Bad examples:

1. Presentation (but not use) of results for the host organization. It is a bad example because it does not guarantee that the presentation has impact (presentation is not the same as using the results).
2. Development of a new tool for the host organization. As long as the tool is not used, it is not an intervention.
3. Analysis of data at the host organization. Just like developing a tool, analyzing the data is not an intervention until the analysis is used at the host organization.

The two lists above indicate the trend that is important to take note of—a good intervention introduces a change. The action team must prepare for such an intervention by establishing a baseline.

The baseline for each intervention has to be related to the intervention itself. The action team must plan for collecting data that can be compared to after the intervention. Since the intervention cannot be undone, the action team can collect the data for the baseline only **before** the intervention.

The data collected during the intervention should also be related to the intervention, and it should be complemented with interviews and observations. The latter are important for capturing such effects that the action team did not plan for. For every intervention, there can be unpredicted consequences, and these need to be captured; the best methods for capturing them are interviews and observations.

Finally, after the intervention the action team must collect data related to the baseline, consequences identified during the intervention, and, finally, the effects of the intervention. The latter must be done in order to prepare for evaluation and learning from the intervention itself. Oftentimes, the lessons learned from the intervention go beyond the initial expectations—that's the power of action research!

Example
In one of the projects, we introduced a new defect prediction model as a means to prioritize testing and development resources [21, 23]. In order to evaluate the model, we collected data about resource allocation before the model was used. We interviewed program managers and integration specialists about

(continued)

their work with resource allocation and defect management. We also collected data about the weekly (and monthly) trends of defect inflow.

During the introduction, we met with stakeholders (program manager and integration specialist) once a week to follow up on their activities and the impact of the action. We showed them the defect predictions and asked whether these predictions deviated from theirs. We also asked about which actions they take to ensure that the defect inflow does not get out of control.

Once the introduction was complete, we collected new data in the same way as the baseline data—trends of defect inflow and interviews.

I often use the following checklist when designing the intervention (yes/no answers only):

1. Is the plan of the organization to adopt the intervention as part of their operations if it is successful?
2. Does the intervention/action make a change in the host organization?
3. Can we collect data (quantitative or qualitative) about the baseline situation before the intervention?
4. Can we be part of the intervention to collect the data?
5. Do we have mandate to adjust the intervention in case of unpredicted events?
6. Can we collect the data after the intervention to observe the effects of the intervention?

If I can answer at least four out of these six questions positively, then I know that I can proceed with the intervention. The other two usually fall into place once I start.

Action research is a cyclic methodology that allows us to repeat series of phases several times. We diagnose the problem, plan for the intervention/action to address the problem, conduct the intervention, collect and analyze the data, and learn from it. Therefore, we often have the possibility to make relatively small interventions in each cycle but accumulate changes over several cycles. In one cycle, we learn from the intervention and plan for new ones. The small interventions in each cycle have a number of advantages; in particular, they do not require radical changes in the host organization. They also allow to adjust or pivot between one cycle and another if we discover and learn new facts.

Teaching Guideline

Since interventions are the most distinct characteristics of action research, we need to spend a significant effort to teach them.

I recommend to have a workshop-based approach to teaching interventions. First, start with the definition of the concept of the intervention, then

(continued)

focus on aspects related to the impact of the intervention, and then focus on how to behave in such a project—explore existing codes of conduct at the companies, conflict management, and expectations management.

This can be achieved by a set of exercises:

- Discuss the definition of an intervention/action—focus on discussing the contrast between making an intervention and studying a case (case study); discuss the difference to participatory observations from the perspective of the impact and bias in the organizations.
- Take up one of the examples of good interventions and one counterexample; discuss their impact on the organization and the scientific community.
- Take up a case of a conflict at a workplace (from literature), and discuss how to handle conflicts; it is particularly important for younger researchers who need this kind of training for their future careers.

7 Ethics of Action Research

The main principle of ethical research should always be to *do no harm*. Our actions as researchers should always have the goal to improve our society and not to harm it. In action research projects, we must protect both the individuals who take part in the research and their organizations. The host organizations invite researchers to become part of them, and they trust them that they will bring no harm to their employees or management. Therefore, open climate, questions, and discussions are the best way to address ethical issues in the action research projects.

Since action research projects are based on interactions with its context and software engineers, we need to provide ethical considerations for the project, in particular, how we will select the participants, how we will store their personal data, and how we anonymize the data so that it does not lead to any harm to individuals and organization.

An interesting aspect is the legal part of the collaborations with the companies. We need to make sure that we have all the agreements in place and that all intellectual property rights are handled according to the regulations specific to the countries where the research is conducted. Avison et al. [3] refer to this as a formalization of the research project.

Selection of participants in our studies must be done using ethical principles. First, we must recognize and understand bias in this process. Although we may be given initial team to work with, we should strive to increase diversity in the project overall. For example, if we have male-only action team, we should find

female colleagues in the reference team to capture a diverse set of view, opinions, and experiences in our research project.

Handling conflicts of interests must also be recognized and monitored during the entire progress of the action research project. Despite the best efforts to be unbiased, there is always a risk of the researcher bias and the Hawthorne's effect. The researcher bias is the inability to objectively scrutinize research results. The action team's stake in the project is often to improve, and therefore, the action team is inherently positively biased and must use reference teams and the management team to counterbalance it [18].

Handling confidentiality can be seen as conflicting the principles of academic freedom and the science's ability to serve greater good. However, in practice, this is not the case. There are two levels of confidentiality—within the company and externally to it. Internally, the team should be open as they are part of contractual agreements. The external confidentiality refers to what we are obliged to keep confidential about the company, its processes, products, and intellectual property rights. We can always generalize our observations, evaluate their validity in other contexts (e.g., open-source communities), and discuss on a general level. We must remember that the scientific community is not interested in so-called "whistleblowers" from the companies, but in new knowledge. This is also what all host organizations are interested in. Whenever we are in doubt, we can always consult company's management, legal team, university's legal team, or even university's ethical boards regarding confidentiality.

Overpromising is also one of the aspects that is important to take into consideration when conducting action research projects [13]. It is easy for the researchers to overpromise to deliver in an action research project and then continue to try to push too much to deliver on that promise. However, the action team should define their deliverables upfront and should be prepared to adjust the plans over time.

Quality of research is sometimes seen as being in conflict to delivering to the company [13]. The action team may feel pressure of delivering new value to the host organization on the cost of quality of research. The quality of research is very cost-intensive as it requires validations, checking for confounding factors and replications, which are often not as value driving as diagnosing, addressing, and validating new problems. Involving other organizations is often much more demanding and time consuming but increases the quality of the research.

In addition to these ethical concerns, there are others that the action team needs to discover themselves. The fact that they are embedded in the host organization means that they are inherently biased and therefore then need the reference team and the management team to discuss and vet ideas and results. I strongly encourage the action teams to discuss potential conflicts of interest based on their specific context.

Teaching Guideline
The best way to teach the ethics of action research is to start from studying ethical guidelines that apply. Every country, company, and university operate in a different legal space, and therefore the action team must:

- Study and understand the contract between all parties in the collaboration
- Identify and operationalize the ethical guidelines applicable, e.g., for using human subjects in case studies
- Engage in discussion about the legal aspects of ownership of research results—include company and university legal advisors

Just by reading, exploring, and asking critical questions about the meaning of clauses in legal documents, NDAs (non-disclosure agreements), and ethical review boards guidelines, we raise awareness of these issues. The students can be asked to identify and explain some of these principles for their project or a project that they find in a literature.

8 Guidelines for Teaching Action Research

Engaging in action research is a very rewarding way of conducting research studies. It allows us to be more embedded in the reality of software development organizations, products, processes, and markets. Getting to the point when the action team understands the intricacies of action research requires training. Teaching action research, therefore, should revolve around learning the following elements:

1. Epistemology of knowledge in action research. First and foremost, new action researchers need to understand what kind of knowledge is produced in action research and therefore what to expect from it.
2. Interventions. Once we introduce the type of knowledge that is created in action research, the researchers need to learn what a good and bad intervention is. They need to understand how the knowledge is produced and therefore what we are expected to do in action research projects.
3. Action team. The interventions need to be conducted by the team, and therefore, the team needs to learn about the roles and responsibilities of each member of the action research. The action team must learn about the differences in roles and the expectations. This makes the team understand how the knowledge is created in action research.
4. Host organization. Once the team understands what kind of knowledge is created and how, in action research, the team needs to understand their obligations toward the host organization and how to navigate the challenging balance between operations and research.

5. Planning and executing action research cycles. Now, the action team is ready to learn about the phases of the action research. Each phase needs to be explained and discussed with the action team.
6. Deliverables. Providing new knowledge has to be done in some form. Therefore, the action team needs to be educated in what kind of deliverables is expected and how to assess if a given deliverable has a good quality or not.
7. Collaboration. Since the action research is collaborative in nature (at least the one that we describe in this chapter), the action team must understand how to collaborate. Models and methods for assessing team maturity are very useful for it, e.g., [11].
8. Ethical aspects of action research. In order to understand what the action team can, could, should, and should not do, the action team must learn about the ethics of action research. They need to understand how to include diverse perspectives in research, how to cause no harm, and how to balance academic freedom with the obligations toward the host company and the research project.
9. Contrasts to similar research methodologies. Finally, the education in action research should include learning about the contrasts to other related research methodologies. The action team must, in particular, understand the differences between the action research and case studies, experiments, and design science research.

Naturally, action research studies can be conducted with different foci in software engineering. Some studies can be more focused on the technology development, e.g., understanding how to use generative AI to create tests, while some studies can be focused on social aspects of software engineering, e.g., how to create a better-performing team. Therefore, it is important to recognize the focus and the context of action research studies and to complement the education in action research with the education in that context.

9 Summary and Conclusions

Action research is an applied empirical research methodology that focuses on direct interventions and actions conducted at industrial partners. The methodology has been successfully used in medicine, nursing, education, and social sciences. However, its adoption in software engineering is just starting to take off.

In this chapter, we learned what action research is and how it is conducted. We have also learned about its pillars— the action team, the host organization, and the interventions. We also learned about the ethics of action research and the way in which we can provide value to both the academic community and the industry. Finally, we finished up this chapter with guidelines on how to teach action research.

As further reading, I recommend to read the newest research studies using action research in software engineering, as this is the kind of research methodology that is on the rise.

Acknowledgments This book chapter is partially sponsored by Software Center (www.software-center.se), which is a collaboration between 5 universities and 17 companies. I would also like to thank Dr. Birgit Penzenstadler and Prof. Yvonne Dittrich for very useful comments on the drafts of the chapter.

References

1. Antinyan, V., Staron, M., Sandberg, A., Hansson, J.: Validating software measures using action research a method and industrial experiences. In: Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering, pp. 1–10 (2016)
2. Argyris, C., Schön, D.A.: Participatory action research and action science compared: a commentary. Am. Behav. Sci. **32**(5), 612–623 (1989)
3. Avison, D., Baskerville, R., Myers, M.: Controlling action research projects. Inform. Technol. People **14**(1), 28–45 (2001)
4. Baskerville, R.: Action research for information systems. In: AMCIS 1999 Proceedings, p. 288 (1999)
5. Çalikli, G., Staron, M., Meding, W.: Measure early and decide fast: Transforming quality management and measurement to continuous deployment. In: Proceedings of the 2018 International Conference on Software and System Process, ICSSP '18, pp. 51–60. Association for Computing Machinery, New York (2018)
6. Dittrich, Y.: Doing empirical research on software development: Finding a path between understanding, intervention, and method development. In: Social thinking-Software practice, pp. 243–262. Citeseer (2002)
7. Dittrich, Y., Rönkkö, K., Eriksson, J., Hansson, C., Lindeberg, O.: Cooperative method development: combining qualitative empirical research with method, technique and process improvement. Empir. Softw. Eng. **13**, 231–260 (2008)
8. Dos Santos, P.S.M., Travassos, G.H.: Action research can swing the balance in experimental software engineering. In: Advances in Computers, vol. 83, pp. 205–276. Elsevier, Amsterdam (2011) .
9. Ferreira, J., Sharp, H., Robinson, H.: Agile development and user experience design integration as an ongoing achievement in practice. In: 2012 Agile Conference, pp. 11–20. IEEE, Piscataway (2012)
10. Lewin, K.: Action research and minority problems. J. Soc. Issues **2**(4), 34–46 (1946)
11. Meding, W., Staron, M., Söder, O.: MeTeaM—a method for characterizing mature software metrics teams. J. Syst. Softw. **180**, 111006 (2021)
12. Melin, U., Axelsson, K.: Action in action research–illustrations of what, who, why, where, and when from an e-government project. In: Electronic Government: 6th International Conference, EGOV 2007, Regensburg, Germany, September 3–7, 2007. Proceedings, vol. 6, pp. 44–55 Springer, Berlin (2007)
13. Morton, A.: Ethics in action research. Syst. Practice Action Res. **12**(2), 219–222 (1999)
14. Ochodek, M., Hebig, R., Meding, W., Frost, G., Staron, M.: Chapter 8 recognizing lines of code violating company-specific coding guidelines using machine learning. In: Accelerating Digital Transformation: 10 Years of Software Center, pp. 211–251. Springer, Berlin (2022)
15. Reason, P., Bradbury, H.: Handbook of Action Research: Participative Inquiry and Practice. Sage (2001)
16. Robson, C., McCartan, K.: Real world research: a resource for users of social research methods in applied settings
17. Sandberg, A., Pareto, L., Arts, T.: Agile collaborative research: action principles for industry-academia collaboration. IEEE Softw. **28**(4), 74–83 (2011)
18. Staron, M.: Action Research in Software Engineering. Springer, Berlin (2020)

19. Staron, M., Medig, W.: Factors determining long-term success of a measurement program: an industrial case study. e-Inform. Softw. Eng. J. **5**, 1 (2011)
20. Staron, M., Meding, W.: Using models to develop measurement systems: a method and its industrial use. In: International Workshop on Software Measurement, pp. 212–226. Springer, Berlin (2009)
21. Staron, M., Meding, W.: Defect inflow prediction in large software projects. e-Inform. Softw. Eng. J. **4**(1), 1–23 (2010)
22. Staron, M., Meding, W., Palm, K.: Release readiness indicator for mature agile and lean software development projects. In: International Conference on Agile Software Development, pp. 93–107. Springer, Berlin (2012)
23. Staron, M., Meding, W., Söderqvist, B.: A method for forecasting defect backlog in large streamline software development projects and its industrial evaluation. Inform. Softw. Technol. **52**(10), 1069–1079 (2010)
24. Staron, M., Meding, W., Tichy, M., Bjurhede, J., Giese, H., Söder, O.: Industrial experiences from evolving measurement systems into self-healing systems for improved availability. Softw. Practice Exp. **48**(3), 719–739 (2018)
25. Wieringa, R.J.: Design Science Methodology for Information Systems and Software Engineering. Springer, Berlin (2014)
26. Wohlin, C., Runeson, P.: Guiding the selection of research methodology in industry–academia collaboration in software engineering. Inform. Softw. Technol. **140**, 106678 (2021)

Action Research with Industrial Software Engineering: An Educational Perspective

Yvonne Dittrich, Johan Bolmsten, and Catherine Seidelin

Abstract Action research provides the opportunity to explore the usefulness and usability of software engineering methods in industrial settings and makes it possible to develop methods, tools and techniques with software engineering practitioners. However, as the research moves beyond the observational approach, it requires a different kind of interaction with the software development organisation. This makes action research a challenging endeavour, and it makes it difficult to teach action research through a course that goes beyond explaining the principles.

This chapter is intended to support learning and teaching action research, by providing a rich set of examples and identifying tools that we found helpful in our action research projects. The core of this chapter focusses on our interaction with the participating developers and domain experts and the organisational setting.

This chapter is structured around a set of challenges that reoccurred in the action research projects in which the authors participated. Each section is accompanied by a toolkit that presents related techniques and tools. The exercises are designed to explore the topics and practise using the tools and techniques presented. We hope the material in this chapter encourages researchers who are new to action research to further explore this promising opportunity.

Y. Dittrich (✉)
IT University of Copenhagen, Copenhagen, Denmark
e-mail: ydi@itu.dk

J. Bolmsten
World Maritime University, Malmö, Sweden
e-mail: jb@wmu.se

C. Seidelin
Novo Nordisk, Bagsværd, Denmark

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2024

D. Mendez et al. (eds.), *Handbook on Teaching Empirical Software Engineering*,
https://doi.org/10.1007/978-3-031-71769-7_15

1 Introduction

This chapter deepens the understanding of action research in industrial software engineering with rich examples and emphasises the practicalities of action research in and with industrial software engineering. The chapter mainly builds on three action research projects that are also used as cases that provide examples to illustrate themes highlighted in this chapter. The first project, The SIM Case, addressed software architecture methods that support the evolvability of a software product that implements hydraulic simulations of water systems (rivers, fresh water and sewer systems). It has been the basis of a PhD dissertation [48] and several articles [46, 49, 50]. The second project, The WMU Case, introduced software engineering and IT management methods of in-house participatory design and user-centred development to an academic capacity-building context [6–8]. The third project, The Data CoDesign Case, explored how domain experts may use data and data analytics to support the innovation and development of vocational education and training [38, 40, 43]. Yvonne Dittrich was the PhD supervisor in all three cases. The cases are introduced in the first three sections of this chapter. However, all the cases have informed all the sections, and we report episodes from various cases to illustrate the theme of each of the sections. All three projects implemented, or were inspired by, the Cooperative Method Development (CMD) approach, which first has been introduced in [17]. Where appropriate, we also use examples from the projects that were the basis of the development of the CMD article.

CMD is an action research approach that combines qualitative empirical research with software engineering tools, methods and process improvements [17]. CMD is a structured methodological framework that is designed to cope with the complexities of action research in an academically rigorous manner (also see the work by Checkland and Howell [12] and by Mathiassen [28]). Five guidelines define the structure of CMD: (1) an action research cycle consisting of three phases (understanding, deliberating change, implementing and evaluating improvements); (2) ethnographically inspired research complemented by other methods, if suitable; (3) a focus on shop-floor development practices; (4) deliberating improvement with involved practitioners; and (5) assuming the practitioner's perspective when evaluating the empirical research and deliberating improvements. Figure 1 illustrates the CMD approach: The empirical research results in an understanding of the current practices that inform the joint definition of one or more problems to be addressed by the action research. Based on this problem identification, an intervention or improvement is developed collaboratively by the researcher and practitioners. The researchers contribute with state-of-the-art research related to the problem identified, based on their survey of the research discourse. Discussions during this deliberation may provide input to the research. Together, practitioners and researchers implement the improvement and evaluate it. This too is informed by research, and it contributes findings and insights that further the research discourse.

The chapter complements Staron's introductory chapter on action research, which lays the foundation for action research and how to teach it. It is presumed that

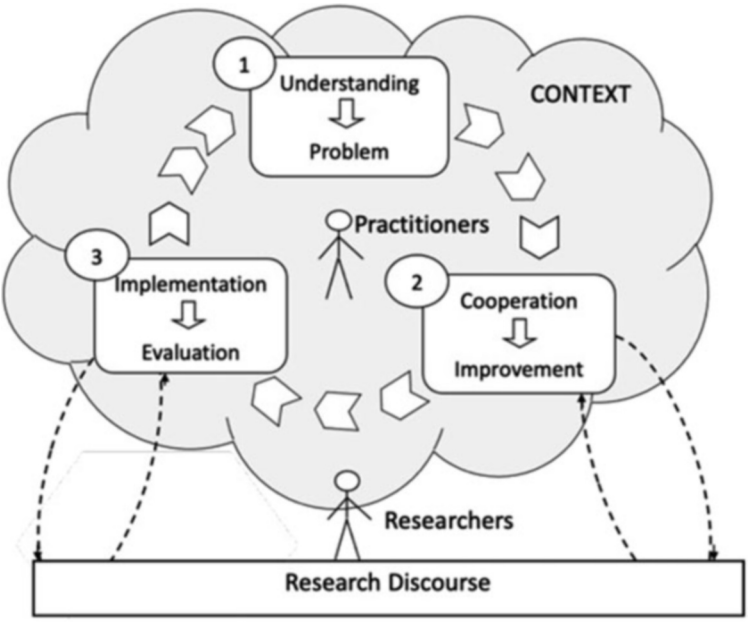


Fig. 1 Illustration of the cooperative method development approach to action research The numbers indicate the 3 phases of the CMD cycle [20, p. 21]

the reader has studied Staron's chapter or a similar introduction into action research before studying this one. It shares the emphasis on Participatory Action Research of Staron's chapter: one of the core principles of participatory action research is the involvement of the practitioners and domain experts whose work practices the interventions address, in the discussion and decisions about these interventions.

However, the CMD approach is more explicit about the choice of empirical research to understand the current situation from a participant's point of view and evaluate the intervention, again using an ethnographically inspired approach. This may be due to its historical development: When we started to apply qualitative empirical research to software development practices in the late 1990s, the companies and software engineers that collaborated with us asked to recommend ways to improve their practices. The CMD approach is designed to support a member's perspective, also when it moves beyond ethnographical empirical research [16], to explore improvements by applying and further developing new methods, processes and tools in an ethically responsible and scientifically accountable way. Similarly, the explicit formulation of this version of participatory action research made our research accountable to the management and software developers with whom we collaborated.

Teaching and learning action research are best done using an apprenticeship model, where the MSc or PhD student, or the researcher new to action research, regularly discusses her involvement with a software engineering team or an open-

source project with the supervisor, a colleague researcher experienced in action research. This chapter reports our experience of learning and teaching action research and may be most useful for an experienced action researcher who is beginning to supervise action research projects in software engineering and his or her students. Throughout this chapter, we will add to a toolkit of methods and techniques that helped us cope with the often-challenging situations we encountered in our research and as supervisors. Thus, the toolkit aims to support learning as it allows to exercise some aspects of action research.

For students and teachers of empirical research courses, this chapter provides a rich set of concrete episodes that supports the discussion of challenging situations in class. Teaching action research through a course presents a challenge for educators, when compared to other empirical research methods, for example, experiments, as it is difficult to practise action research at small scale in the classroom. Therefore, in our courses, we ask students to formulate a research proposal for an action research project that is connected to their main research focus, as a way to apply what they hear and read about. Alternatively, the teacher could use the course improvement as an example for the whole class. You will find a number of exercise proposals in the remainder of this chapter. They either ask the students to apply the lessons learned in a section by enhancing their project proposal or implement some of the tools and techniques in the classroom.

We start with general considerations of ways to anchor action research in the organisation and address the challenges of not only understanding the work practices of the practitioners whose participating in the research but also the organisation and the internal and external stakeholders. The toolkit provided in Sect. 2 discusses the formation of a steering committee, the use of ethnographic field work to understand the work practices of the practitioners involved and explore the challenges from a practitioners' perspective, stakeholder analysis methods to map the wider organisational context and the formulation of an informed consent form for the participants. The exercise asks you to apply some of this knowledge to the (fictional) project that is presented as part of Exercise 1. Two cases are introduced in Sect. 2, and the third case is introduced in Sect. 3. Section 3, Deliberation: Anchoring Interventions and Improvements in the Organisation, is especially relevant to *participatory* action research. We have added an introduction to how to design, implement and make use of workshops to the toolkit. The exercise is designed to allow you to practise designing and implementing workshops. Section 4 discusses the use of visualisations as a potent way to foster discussions. The visualisation section adds to the toolkit the presentation of rich pictures, which we use in all our projects, and pointers to additional visualisation techniques. The exercise invites you to add a visual interpretation of the stakeholder analysis of your project proposal. Section 5 takes up what may be the most difficult part of action research: how to handle situations where things do not work out as planned and discuss with the practitioners. This section adds peer debriefing to the toolkit, and the exercise invites you to experience a debriefing situation. Section 6 discusses how action research interventions can lead to sustainable change in the collaborating organisations. The toolkit presents a framework for analysing and discussing the

sustainability of change in the organisation. The exercise invites you to discuss the interventions you plan for your project and to consider how to improve their organisational sustainability. Section 7, From the Action to the Research Results, concludes this chapter and highlights ways to ensure that the action research findings and insights may be supported by a scientifically accountable documentation. It adds ethnographically rich points and thematic analysis to the toolkit. The exercise asks you to explore the kinds of contributions you may expect from the research project you developed through part of the exercises.

Exercise 1
Together, some of the exercises in this chapter result in the (improved) design of an action research project proposal. Staron does not propose to do a project proposal.

If you already plan to do action research, feel free to use your project and the proposal you probably developed as a starting point. If you use other methods in your research, try to find an angle to your research that could be executed as action research. For example, if you work with software architecture notations, a related action research project might involve collaborating with a team and seeing whether you can jointly improve their architecture practices. Another possibility would be to pretend that improving the course is an action research project and use that fictional action research project as a basis for the exercises.

If you are not building on an existing research proposal, develop a short motivation for your study (about half a page), and formulate a research question related to action research. Describe the (fictional) company and team you plan to collaborate with or—if you decide to use the course improvement as an exercise—a (fictional) university whose research methods course or PhD programme you would like to improve.

2 Action Research in the Software Industry

Action research in software engineering projects, for private or public organisations, is potentially rewarding but challenging. It provides an opportunity to make improvements by contributing knowledge and development capacity that are unavailable in the setting where the action research takes place. It is also an opportunity for the action researcher to learn about real-world operating environments and the usefulness of the technologies being developed. At the same time, change in organisations is difficult, and there are many stakeholders with different and sometimes conflicting priorities that need to be addressed. These stakeholders include other software engineers, users and managers. The subsections further

discuss what must be considered when starting an action research project with a software development organisation.

Case 1: SIM—Evolvable Software Products for Hydraulic Simulation
Background The research collaboration with SIM was intended to explore methods, tools and techniques, to ensure the evolvability of software products. SIM develops software that models one-, two- and three-dimensional bodies of water, to predict the effect of, for example, dam construction. The software was further developed to simulate near-real time predictions, to support water management. The researchers collaborated with the team responsible for the product that simulated open one-dimensional water systems, such as rivers and creeks. As part of the re-engineering, the company decided to merge this software with a sister product, which facilitated the simulation of closed one-dimensional water systems, such as sewers.
Action Research The action research applied the Cooperative Method Development approach [17]. The action research introduced lightweight software architecture techniques for high-level design, developed a lightweight Architecture Level Evolvability Assessment for focussed discussions of design decisions with relevant stakeholders and introduced lightweight architecture compliance techniques that use the built system. The research results emphasise the need to adapt software architecture methods and tools to support the continuous evolution of software products: architecture design and evolution takes place as part of everyday software development; architectural practices need to help the software architect to keep up with the changes to the software and the emerging requirements that may challenge the architecture; evolvability is a quality that should be considered during regular software architecture design discussions [46, 47, 49].
Anchoring the Research in the Organisation This project was part of a larger project for the design of evolvable software products, in which another company was also involved. The contact was established by the university's senior management. The head of software development was the management representative for the joint project's steering committee. Locally, the tech lead of the open, one-dimensional simulation product was the main contact point.

The research collaboration was part of a broader development to professionalise software development in an organisation where up to that point, the software had been regarded mainly as a tool for the main business, the consultancy that used the software to model and simulate changes to water systems.

The parallel professionalisation affected the project later: Halfway into the 3-year project, the company decided to reorganise. Prior to the reorganisation, the development of the software products was placed with the departments

(continued)

that used the software for consultancy. The reorganisation positioned the software development of various products in an own department. However, this also resulted in the tech lead and main contact leaving the company.

2.1 Anchoring the Action Research Project in the Organisation

Action research projects aim to change how organisations develop or use software. Even if the research addresses only the development of one project, such a change may affect the whole organisation and may have an impact on the organisation's business and economic outcomes. Therefore, it is important to anchor the action research project in the organisation's management and development. Management needs to be on board with the research and its outcomes. The development team that collaborates with the researchers, likewise, needs to feel secure in the fact that they are not expected to implement methods that force them to act against their better judgement.

At first glance, the anchoring in the management may appear to contradict the above-mentioned participatory approach to action research. Therefore, when anchoring the project in the organisation, the commitment to the bottom-up improvement needs to be negotiated and agreed on by all parties. This is not only an ethical concern; it also allows the establishment of a situation where the methodological interventions may be based on professional requirements, rather than on (anticipated) management opinion [16].

As the SIM case above indicates, the relation between the researchers and the organisation needs to be maintained for the duration of the project. During the 3 years that an externally funded project in Europe lasts, changes to the organisation are to be expected. Similarly, participatory action research often develops its own dynamic, and the final outcome may not be what was anticipated by either the company or the researchers involved. Therefore, the dialogue between researchers and the organisation needs to continue throughout the project. One way to ensure and structure an ongoing dialogue is to establish a steering group that includes representatives from both research and practice (see tool box).

At the beginning of a research collaboration, the main purpose of a steering group is to formulate shared goals and align the expectations of the collaboration. The nature of the steering group will develop over time, as the research project matures and the researcher(s) become more integrated into the organisation. Throughout the research project, the steering group is important for ensuring that the research has the necessary organisational backing. Especially, all interventions decided together with the participating team also need to be discussed and agreed on by the steering committee. Also, if other stakeholders like business departments or intended users of the software subject to the collaboration should be involved, the steering committee

has to secure the organisation's necessary support. Towards the end of the research collaboration, the steering group is an important forum for ensuring that shared goals are met and the knowledge gained is shared.

Case 2: IU—Domain Experts Co-designing Data

Background The research collaboration with Industriens Uddannelser (IU) was intended to explore how domain experts (who were not IT professionals) could participate in the design and innovation of the data and data structures that underpinned the data-based services they used, as well as provided to other stakeholders, in their work practices.

The research project originated in the quest to address the societal challenge of advancing small and medium-sized organisations' capacity to develop and support ways of innovating and designing services by using data more intelligently. IU develops IT systems that support the maintenance and development of vocational and continuing-education programmes related to the industrial sector.

Action Research The researcher was heavily involved in the organisation: present at the organisation at least 3 days a week and participating in many internal meetings and other social activities that extended beyond the scope of the action research interventions. The researcher collaborated with several groups at the organisations (see [24] for an appraisal and comparison of the project as a long-term action research project that targets infrastructure systems).

The project applied an action research approach as a 'meta-practice' [22] and as a process of critical inquiry. The action research was also inspired by Robson and McCartan [37] and followed the widely used representation of a spiral or cycle, where each intervention involves three general stages: (1) planning a change, (2) implementing the change and observing what happens following the action(s) and (3) reflecting on the processes and the observed changes to plan for further change and the continuation of the cyclical process (see Fig. 2). Three cycles were implemented.



Fig. 2 Case 2: action research process [38, p. 27]

(continued)

Anchoring the Research in the Organisation Industriens Uddannelser contacted the university to obtain support for making better use of various data sources that were available to the organisation. As the organisation was co-owned and financed by labour unions and industrial interest organisations, the management organised a workshop to anchor and scope the research project, even before the application was started.

The first research activities focussed on understanding the organisation's various administrators' current ways of working with data. A natural contact for the researcher was a group at the organisation that provided statistics to various committees and working groups. However, other members of the organisation were also involved in the action research cycle.

Throughout the project, the management of Industriens Uddannelser supported the project by allocating resources and time to help the researcher, enabling the co-design of methods with relevant members of the organisation, and by anchoring the project with the external stakeholder.

2.2 Understanding the Problem Before Implementing Changes

The first phase of all action research cycles consists of doing empirical research before any intervention takes place. One of its purposes is to be able to document the change brought about by the intervention. If we take the SIM case as an example, to document that the new software architecture and architectural practices are supporting the evolution of the software product in a better way than the existing software architecture and architectural practices, the latter need to be understood and documented. However, the research should not just document a benchmark to compare it with the situation after the intervention. The initial field work, in particular, should be rather broad and take a flexible approach [37], to understand how the software development takes place and the rationale behind the existing practices. We recommend using an ethnographically inspired approach that emphasises the understanding of a social situation from a member's point of view: understanding software development from a practitioner's point of view and the rationale behind practices that may appear idiosyncratic at first glance will help the researchers to observe and assess both the intended impact of the intervention and the unintended side effects.

One example for the importance to take the practitioner's perspective seriously is related to the SIM project: the project and the tech lead were reluctant to use architecture documentation. Instead of attributing this to their lack of software engineering training, we conducted an interview study to see what other software product teams do, in terms of architecture documentation [50]. The results helped us carefully adjust the software architecture practices in order to support communication, rather than replace it [46].

The initial empirical research is also needed to later customise the intervention for the specific organisation. It provides the basis for understanding what other not-yet-considered factors influence the applicability of specific methods. Referring to the research above: prior to our study, the research on software architecture documentation did emphasise the communication of this architecture to the developers, but did not reflect the effect of a software architecture document on the communication from the software development team to the architect, when they evolved long-living software products. Without prior broad empirical research, we probably would have ignored the team's reluctance to adopt the use of architecture documentation.

Toolkit Anchoring
Steering Committee
To ensure that all parties involved are regularly informed and to ensure the necessary resources for a project, the establishment of a steering group for the action research project is crucial. The steering group should meet regularly to discuss the progress of the research and the change initiated, to sanction on the next steps, and to assure the necessary resources in form of time of the development team, access to relevant documents and support for additional research activities.

The steering group should consist of a representative of the management of the collaborating company; a representative of the development, such as the development project's project manager, with whom the researcher collaborates; the researcher who is implementing the action research; and, if the acting researcher is a PhD student, the supervising professor. The researcher or supervisor would also typically stand for the anchoring of the project with the university. Thus, the steering committee links the actual day-to-day research with management on both sides, software development organisation and university. The steering committee should be able to take relevant decisions regarding the action research project.

In addition to handling the contractual side of the project with respect to confidentiality, intellectual property and publication procedures, the steering group should also agree on the principal lines of the action research and, especially, the participatory character of the action research. For the team to freely discuss and adapt methods and tools, they need to be sure that a critical attitude to management's preferred methods does not adversely affect team members. This also means that the researchers need to negotiate with the organisations' management that they do keep the team's confidentiality and communicate the empirical findings only after they have been cleared with the team.

(continued)

Ethnographic Field Work
Ethnographic studies are the subject of this volume's chapter by Dittrich, Sharp and de Souza. We recommend reading that chapter and doing the exercises provided there, to plan the ethnographic field work at the beginning of an action research cycle.
Stakeholder Analysis
Stakeholder analysis is a well-known way to systematically map relevant groups of actors, to consider their interests, for example, when managing a software development project. In strategic management, a number of techniques have been developed to identify stakeholders and analyse their interests and relevance. The article, 'What to do when Stakeholders matter. Stakeholder Identification and Analysis Techniques' [11], provides an overview of the most widely used methods. Bryson offers various definitions of the stakeholder: in one of them, the stakeholder is defined as 'any group or individual who can affect or is affected by the achievement of the organisation's objectives' [21, p. 46]. When applying stakeholder analysis techniques to a research project, the term 'research project' replaces 'organisation'. Two of the techniques presented could be a starting point for thinking systematically about stakeholders: the basic stakeholder analysis technique and the power-versus-interest grid.
Basic Stakeholder Analysis Technique The basic stakeholder analysis technique [10, p.71–75] is a structured brainstorming method that may be used by individuals or groups. The process below is an adaptation of Bryson's proposal [11, p. 29–30].
Brainstorm the list of potential stakeholders:

- Prepare a separate flip-chart sheet for each stakeholder.
- Place a stakeholder's name at the top of each sheet.
- Create a narrow column down the right side of each sheet, and leave the column blank.
- For each stakeholder, in the area to the left of the narrow column, list the stakeholder's expectations of the action research project.

Decide how satisfied the stakeholder is with the (planned) project. Use coloured dots to indicate a stakeholder judgement of 'good' (green), 'fair' (yellow) or 'poor' (red) in the empty right-hand column of their flip-chart sheet.

(continued)

- Identify and record what may be done to quickly address a stakeholder's concerns.
- Identify and record longer-term concerns with individual stakeholders and with the stakeholders as a group.

Additional steps may be included, such as:

- Specify how each stakeholder influences the project.
- Decide what the project needs from each stakeholder.

Power-Versus-Interest Grid The power-versus-interest grid was first described by Eden and Ackermann [18]. Figure 3 presents the general idea:

The methods may be implemented by drawing the grid on a whiteboard and using sticky notes to position stakeholders, and move them according to the discussions of their interests in the project and their ability to influence the project, for example. You may use the stakeholders identified in the previous technique, or if you start with the grid, you may brainstorm who the relevant stakeholders might be.

Both techniques give an idea of the organisational and inter-organisational setting for the action research project.

Informed Consent

Informed consent is a widely used concept in research with human subjects. It means that the people recruited for research need to agree to be subject to the research based on information that allows them to assess possible negative impact of the research on themselves. The informed consent can take many different forms. It can consist of presentations of the research that are signed by the project participants. It can also consist of the oral information about how data is handled in the beginning of the interview together with the assurance that the interviewee can ask the researchers to destroy the recording at any point in time. What form is adequate needs to be decided from case to case.

In our practice, we have good experiences with written text containing a short description of the project, a short description of the collaboration, a short presentation of the university and the research group maybe including names and background of the involved researchers, possible benefits for the participating developers, how the research might take place (observation, interviews, workshops), how the data is handled and contact information of the researcher and, in case the researcher is a PhD student, the supervisor.

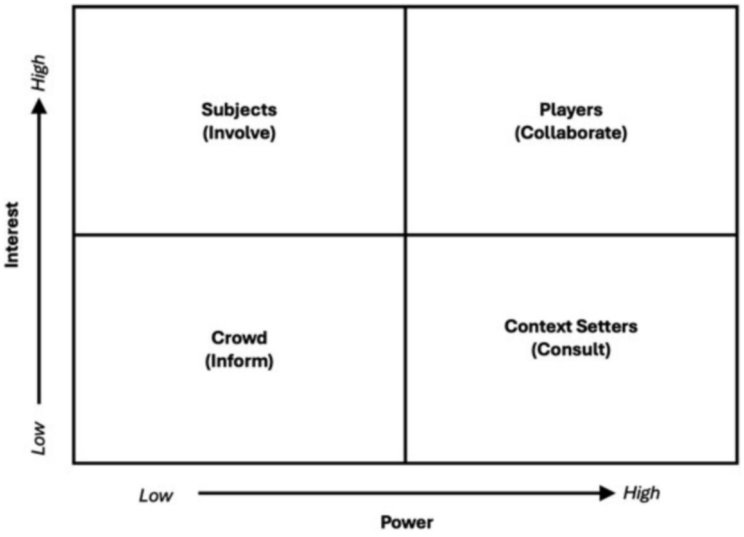


Fig. 3 Power-versus-interest grid [18]

2.3 Responsibility and Interventions

Action research does not aim to only understand software engineering practices; it also aims to intervene in and change software engineering practices. For many organisations today, software is either the main product or the core infrastructure of their business. In addition to ethical considerations that apply to other qualitative research, the deliberation surrounding and implementation of change and interventions need to be carefully considered and executed responsibly.

As in ethnographic research, the impact of the disclosure of individual actions, the development team's software practices and how software is developed in a company may have repercussions for the individual, the team or the company as a whole. As with qualitative research, the researchers need to first clear the citations and references to the work practices with the individual and team, before research findings are disclosed to management. Similarly, the company must be able to request that the research publications do not mention the company or that they hide specifics of the project (for a more comprehensive discussion of this dimension of research ethics, see the 'Teaching and Learning Ethnography for Software Engineering Contexts' chapter by Dittrich, Sharp and de Souza).

Likewise, if an intervention adversely affects an organisation's ability to develop software, it may have a severe impact on economic outcomes. Therefore, the researcher's interventions need to be carefully deliberated at both the team level and the company management level. The company and the team need to have the final say, as the researchers cannot assume the responsibility for financial losses. This also implies that informed consent forms for action research projects not only need

to include information about handling data but also must detail how the interventions are deliberated and implemented. Similarly, the steering committee needs to discuss both the informed consent and how interventions are chosen and implemented.

The research process may have effects on the organisation. Participatory action research emphasises the involvement of the people or team whose work practices are the subject of the intervention. This also needs to be anchored in the company management: changes that might affect the organisation are often ordained by management; when engaging in action research on software engineering practices, the company management must be informed of, and agree to, the relevant software teams also being involved in deciding about the intervention. This needs to be discussed and agreed on from the outset. The empowerment of the software team may in turn lead to internal tensions. For example, other teams may envy their opportunity to experiment with new methods. Similarly, the interventions may be seen as infringing on other organisational stakeholders' interests.

Historical developments in a company also need to be considered. A Danish project that was collaborating with various companies on CMM-based software process improvements found that in one company, previous software process improvement projects led to so much resentment among the company's software developers that the research team had to switch to a problem-based approach to software process improvement [23]. The next section focusses on the participatory deliberation of the intervention, as part of the action research.

Exercise 2: Anchoring
Extend the text developed for Exercise 1 with the following:

1. A proposal for the team to collaborate with and a proposal for the steering group
2. A plan for the initial research aimed at understanding the current practices
3. A stakeholder analysis that focusses on the possible effects of changes to the aspect of software development subject to the proposal
4. An informed consent document that you may either share with the individual developers or present to the team, for example, as one of the slides that introduce you and your research

3 Deliberation: Anchoring Interventions and Improvements in the Organisation

One of the core considerations of any action research project is the deliberation of the intervention. Especially when collaborating with industrial and public partners, the responsibility for the decision needs to remain with the organisation: changes to a company's software development practices may have a substantial impact on

the company. In the worst-case scenario, the quality of the software developed or the operations based on the software could jeopardise the company's existence. Therefore, both the CMD approach [17] and Staron's chapter, which introduce action research, emphasise involving relevant parts of the organisation in the discussion and decision on the changes. This deliberation often takes place at two levels: when a collaboration between a university and an industry begins, the researchers involved, representatives from the company and practitioners agree on a general research focus. When the partners will be applying for external funding, this will be part of the joint application.

Once the project has started, the general focus will be further developed: the initial empirical research may refine the understanding of the problem for both the researchers and the software developers involved. In parallel to this, a literature study by the researcher may yield additional ideas for how to address the organisation's needs. These results should inform the discussion of and decisions about the interventions.

The deliberation needs to bring together the results of the initial research, an agreement on the (refined) problem to be addressed, various possibilities for addressing the problem and a decision regarding what and how to implement the change that is expected to address the problem. The presentation of initial research results should involve a discussion by the team members, regarding whether the team can confirm the results and, especially, the identified problems and challenges.

Sometimes, for example, when the initial research is comprehensive and has yielded new insights or when various kinds of interventions need to be explored through further research, it may be necessary to organise a series of workshops, to give the researchers time to prepare their findings and input and the participants time to reflect on the discussion. If the intervention prioritised by the project is mission critical, further exploration, and perhaps even experimentation, may be needed, before a decision may be taken. The text box below presents an example of such a complex deliberation process.

Deliberation Example: Personas Is Not Applicable

The article 'Personas is not applicable: Local Remedies Interpreted in a Wider Context' [35] presents the lessons learned from a long and complex deliberation process. The initial research with the interaction design team of a mobile operating system and application developer identified a lack of ability to communicate the empirical grounding of interface design decisions. Personas were identified as a way to connect and discuss interaction design in relation to an abstract representation of the results of the empirical research and also allow to bring in concrete user research data into the discussion.

The researcher did a literature survey and presented the academic perspective on Personas, which at the time of the research was a relatively

(continued)

new approach. The Interaction design team developed relevant personas to illustrate this approach. At the same time, Personas were explored through student projects at the university. The interaction design team decided on the Personas approach. However, to apply this approach, the team needed to convince the software development organisation and Marketing and Sales. The software development team and the executive management were brought on board. However, Marketing and Sales wanted to involve the customers. The customers consisted of competing mobile phone producers who teamed up to share the cost of operation system development. They could not agree on a common set of personas. In the end, this resulted in closing down the Personas project.

The analysis of the deliberation process resulted in the insight that a business setting might lead to that approaches that are fully valid from an interaction design and software engineering perspective are not applicable.

The example above also shows that the deliberation process has to take place at the team level and must be anchored in management. Here, the steering committee discussed in the previous section plays an important role. The manager of the steering committee needs to be able to pinpoint which stakeholders to involve in the decisions in an intervention.

It is crucial for the researcher to understand that deliberation is not just about coming to a decision on the intervention but that the discussions themselves are research data and often provide an opportunity for a deeper understanding of the practices to be supported and aspects that may influence the applicability of certain methods. For example, in the hydraulic simulation case, the practitioners' reluctance to develop comprehensive documentation of the software architecture resulted in a broader, interview-based study of software architecture awareness [50]. The interviews with tech leads and architects of a number of very diverse software products showed that software-product architects need to be informed of the developers' changes, to guide the software developers and to be up to date on the challenges. In turn, these findings reveal an under-researched and neglected part of software architecture.

As action research takes place in cycles, the results of one intervention may lead to the identification of new issues and influence the decision on the next intervention. The following WMU example provides further insights into how multiple action research cycles developed during the research process. The example shows that the action research could not be planned ex ante but that the researcher's and the organisational stakeholders' learning about needs and deliberating solutions in one action research cycle prompted new research cycles.

Case 3: World Maritime University (WMU)—Infrastructure and Methods for Participatory and End-User In-House Development

Background The action research was initiated to support shop-floor IT management practices at the World Maritime University (WMU). WMU operates with a capacity-building mandate under the United Nations special agency, the International Maritime Organisation (IMO). The shop-floor IT management practices were characterised by a close-knit collaboration between domain experts and IT professionals. Various shop-floor IT collaborative management groups were involved internally, to develop organisation-critical ICT systems, but were challenged by the need for an integrated infrastructure. The action research included three shop-floor IT development practices, including a Learning Management System and a Student Life-Cycle Management system to support the organisation's capacity-building and educational programmes, and an organisation-wide contact database and an electronic forms system. Their composition, size and scope of development changed throughout the study, as needs and solutions evolved through action research.

Action Research The Cooperative Method Development approach [17] guided the action research and was adapted to develop organisational IT management structures and processes, together with the organisational actors and IT professionals involved. The action research was a long-term project and was carried out in three overlapping cycles, connected through their phases of understanding needs and deliberating and evaluating improvement.

Anchoring the Deliberation in the Organisation The various shop-floor IT collaborative management groups gradually became part of the research. The aim was to understand the needs and support improvements from the point of view of the shop-floor design constituencies that managed IT in the organisation. For example, in the first action research cycle, the researcher joined a team of faculty support staff to develop a new Web-based scheduler. This yielded situated insights into how a technical platform enables and constrains application development and also how it is possible to work with participatory tools and techniques with the users. For example, a technical platform that supports only custom development had fewer provisions for rapid prototyping and end-user development. These realisations indicated that the technical platform could not be black-boxed for the users, but they needed to participate in its design. In the next action research cycle, the Student Life-Cycle Management system and contact database were included in the action research. Here, the focus was on how participatory tools and techniques could be used to involve users in the design of a new, organisation-wide Enterprise Resource Planning (ERP) system.

A range of various PD tools and techniques was applied, including functional analysis, to acquire an initial overview and gather requirements

(continued)

[3]; participatory observations (and stakeholders learning through overtaking work functions), together with story card summaries to acquire an in-depth understanding of current work practices [3, 5, 26]; stakeholder workshops that used rich picture collages to identify key work practices and integrations and also included mapping new solution scenarios; and company visits, presentations by vendors, reviews and experimenting with prototypes [3]. The results showed how it was possible to involve the users in the design of the new ERP system and how the PD tools and techniques supported the reorientation of shop-floor IT management practices, which were prompted by the technical and organisational integrations required by the new system. Thus, the results showed how it was possible to continue to empower users and apply shop-floor IT management practices when designing an integrated IT infrastructure. The final action research cycle targeted organisational IT management practices and how users could jointly make decisions about their IT infrastructure and specific development projects through an IT steering committee. Improvements included both planning and decision-making that were connected to shop-floor IT management practices. The focussed action research cycles were not decided ex ante but gradually evolved according to organisational needs.

The intertwined recognition of needs and deliberation and the exploration of interventions were closest to each other in the Industriens Uddannelse case. The example below describes the close collaboration around the exploration of, and experimentation with, various representations and methods to support the use of familiar and new data sources.

Deliberation Example: IU—Anchoring the Deliberation in the Organisation

In the IU case, deliberation and intervention were often tightly intertwined; workshops that explored and designed tools and methods revealed the need for additional support, which was then addressed in a new workshop. An example is the third intervention of the IU project, which aimed to combine the learning from the previous interventions that addressed existing data practices and building design capabilities at the organisation, to further explore how the organisation could explore new data sources and experiment with their usefulness. The intervention included a workshop series in which the researcher explored how members of the organisation could handle various data representations and their ability to co-design data. For example, one workshop focussed on how the members of IU could create ‘data searches’ as

(continued)

a way to explore how they look for data. Together, these workshops explored ways of 'zooming out and zooming in', an approach [32] to collectively understanding existing data practices. This combination of 'macro- and micro-levels' demonstrated that the education consultants worked primarily with existing data sources that were 'ready at hand' and made only limited use of data in new and innovative ways.

This exploration motivated the implementation and co-design of IU's Data Sphere (Fig. 4), which was a tool that aimed to encourage all members of the organisation to consider and generate ideas for new data sources that the organisation could explore. The intervention revealed an inherent focus on data, which supported the organisation's consideration of data as something that may be used to innovate.



Fig. 4 Case 2: Datasphere

The preceding examples show that, in many cases, deliberation consists of a series of activities. We recommend that beginning the process with a presentation of the results of the introductory research. This first step should give time for feedback from the team. A result might be a list of concerns or pain points that the team prioritises through some voting mechanism. The next step could be the researcher presenting one or more approaches to addressing the pain points, which again may be subject to prioritisation.

As mentioned above, apart from the members of the development team, the management of the company's development organisation also needs to be involved. Depending on the company culture, the agreement with the development team and its availability, management may take part in the deliberation workshops organised for and with the team. If that is not the case, or if ethical considerations are against it, the team's decision needs to be confirmed by the steering committee. If necessary, the team and the steering group may involve other organisational stakeholders. Here,

the initial mapping of relevant stakeholders and the initial research must inform the deliberation process.

The workshops in which the action research team discusses interventions, among themselves or with other parts of the organisation, need to be carefully prepared and implemented. The Toolkit Deliberation presents how to prepare, implement and document workshops to support the practitioners involved and gather the data needed to support the research publications. Workshops may also be organised to fit the project and company culture. For example, when collaborating with an Agile team, the researcher could use one of the retrospectives as a deliberation workshop. Retrospectives are a construct for Agile development teams to reflect on their practices and to decide how to improve them. Similarly, project post-mortems may be used to jointly reflect on implemented improvements and evaluate them from a team perspective.

Toolkit Deliberation

Workshop Preparation, Implementation and Aftermath

Especially in a software engineering context, the researcher collaborates with teams that consist of several people, and often a wider range of stakeholders is also involved, so workshops are often a good way to moderate a discussion among relevant stakeholders and prepare a decision.

Inspiration for workshops may be drawn from co-design and service design methods: In co-design and service design, the software, the collaboration among domain experts and the use of software applications are the subject of design. When researching software engineering, changes in work practices of the software team or the application of some methods and techniques are subject to the research.

Workshops need to be prepared, in order to foster useful discussions and feel effective and efficient to the participants. As the emphasis is on practitioners' discussion and the input, presentations should be kept to a minimum.

Developing and leading a workshop is a skill that needs to be practised. We recommend the following process:

Preparation You need to decide the purpose of the workshop and the intended outcome. If we use the improvement of a PhD study programme as an example, one goal of a first workshop could be identifying and agreeing on the core concerns and priorities. The next step would be to identify what needs to be in place for the participants to achieve this outcome. Next, activities should be selected that help achieve these outcomes. This process may be recursive; especially when an activity itself requires other prerequisites. In the example of PhD programme-improvement, we could choose to brainstorm challenges of PhD studies and cluster them. Brainstorming could then be followed by a

(continued)

discussion of subgroups, which might lead to the elaboration of the root cause and the effects of the concerns. When the concerns are elaborated, this could be put to some form of vote.

The foregoing activities should be further refined and developed into a playbook, best presented as a table (horizontally formatted) with columns for time frame, activity, purpose, person responsible, approach and prerequisites. The time frame should be broken down to 5-minute intervals, where appropriate.

In parallel with the planning, time and place need to be chosen: A room needs to be reserved. Especially for longer workshops, coffee and catering need to be organised. Necessary materials, such as adhesive notes, paper and pens and the like, need to be acquired.

Implementation Usually, the workshop moderator would not be able to take field notes. If possible, workshops such as the one developed above should be audio- or video-recorded, as people may point or otherwise communicate non-verbally.

Other group members or the supervisor might participate in the workshop, to support the researcher in case of various contingencies. For example, the workshop dynamics may develop in a direction that requires a deviation from the play book. Here, a second participant might help manage the situation. If it is not possible to record the workshop, they could also take notes throughout the workshop.

Aftermath Before leaving the room, recordings need to be stopped and secured. The results of the workshop need to be documented: whiteboards and other physical results need to be photographed; if electronic media such electronic whiteboards were used, a screenshot needs to be taken. After the workshop, the room needs to be cleared according to the organisation's guidelines.

Soon after the workshop, the participating researchers might meet for a debriefing, to discuss noteworthy events during the workshop and unexpected developments. The implications of the workshop's outcomes for the next steps of the research need to be discussed.

Exercise 3: Planning and Implementing a Workshop

For this exercise we use your research method course as a case. Plan a mid-term course evaluation using the Tool Kit Deliberation. Work in smaller groups and develop a playbook.

The playbooks can be presented in class and the class could vote on a specific playbook.

(continued)

The team may then carry out their mid-term course evaluation using the chosen playbook.

4 Visualisations: Supporting Mutual Understanding and Materialising Complexity and Change

Action research often involves various forms of domain expertise. Apart from the researchers involved, an intervention might include other researchers, stakeholders with deep business knowledge, stakeholders with shop-floor expertise and software engineering stakeholders. These various domain experts bring valuable knowledge to the project but might lack a shared language, which can lead to misunderstandings. Therefore, it is important to establish ways for the involved parties to develop a mutual understanding, to allow an intervention to develop and (ideally) generate the intended change. One way to go about this is by creating visualisations.

Visualisation Example: WMU—Establishing Mutual Understanding Through Rich Pictures
Rich picture visualisations were an integral participatory technique when a design proposal for an Enterprise Resource Planning (ERP) system at WMU was being developed. An ERP system is an integrated infrastructure that connects an organisation's administrative systems. Because of the complexity of connecting various technical systems and their associated work practices, ERP systems are difficult to design and implement [13]. In the WMU case, the challenge was to support the users' participation in the design process. The focus of the second action research cycle was to understand how rich picture workshops could be used in combination with other participatory design tools and techniques, for this purpose [3]. The action research cycle started with participatory observation.

(continued)

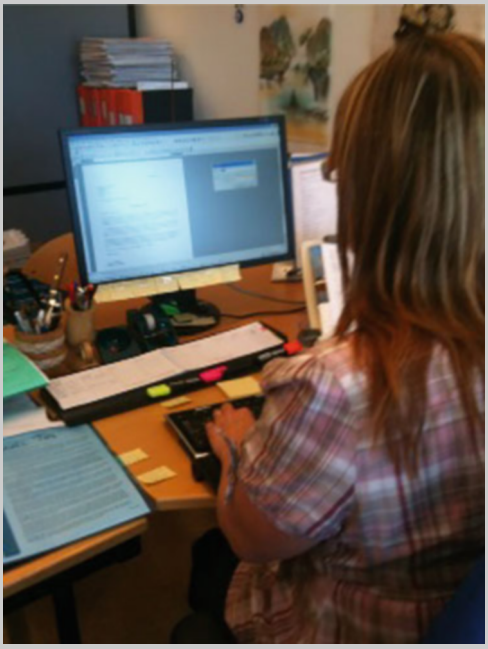


Fig. 5 Case 3: Observing domain experts

The example of the image in Fig. 5 shows how the researcher observed one of the administrative assistants, to gain an in-depth appreciation of her work, and how she worked with the Student Life-Cycle Management system. The insights formed the basis for documenting the work practices and use of system support in what may be referred to as story-card summaries [5]. The outcome of the story card summaries from various departments was then visualised as rich pictures, as seen in the Fig. 6.

(continued)



Fig. 6 Case 3: Extended rich pictures

These rich pictures were first used so users from various departments could learn about each other's dependence on system support and how various systems were linked. Following this, the rich pictures were used to develop new ideas for improved integration, and workshops were arranged so the users and the researcher could redraw possible connections between the systems. The outcome of the rich-picture-based design work was summarised in new story card summaries, and ERP system service providers were requested to use them as the basis for their presentation so the users could follow these presentations, based on their previous collaborative work.

Visualisations are valuable artefacts in action research, from both the research perspective and the change perspective. From the research perspective, visualisations can support and deepen the researcher's understanding of the industrial setting. Examples of such visualisations include representations of the organisational structure, the broader framework of the organisation or the IT infrastructure. The (often) concrete format of visualisations allows for instant 'member checking' as it is required for quality assurance in qualitative research [37]: for example, the researcher may ask members questions about the organisation, about their understanding of a visualisation and repeat these in situ interviews parallel to the ongoing adaptation and development of the visualisation. Moreover, a visualisation may act as a tool for reflection that supports the researcher's and the organisation's understanding of the often-complex reality which they must navigate, to achieve the intended change. From the change/learning perspective, visualisations may materialise the (sometimes) invisible action research intervention. Visualisations may help demonstrate the impact of the intervention. Finally, if a visualisation is developed over time, it may also act as a way to document the development of the intervention.

Visualisation Example: SIM—Code Comparison Triggering Integration of Two Software Products

As part of the field work, the action researcher at SIM analysed the code of the software product subject to the collaboration. Part of that analysis was also a comparison of two sister products that simulated open and closed, one-dimensional hydraulic systems. As quite a few people became interested, the researcher displayed the visual representation of the results of this comparison in the company's hallway. Shortly after, she observed developers of the two products standing in front of her printout and discussing the (minor) differences. The visualisations of the similarities and differences between the two products prompted a discussion that ultimately resulted in the integration of their computational core. Though the code analysis was a result of the researchers' attempt to understand the architecture of the software, the visualisation also influenced the decisions to re-engineer both products.

As foregoing the examples show, the use of visualisations differs from traditional system specification. Their difference may be understood as the difference between the intention to define and to remind [26]. Visualisation and participatory tools and techniques do not stand independent from the situation in which they have been produced but are intended to remind (not define) the design team of work situations in particular need of computer support. They become boundary objects [45] that establish a field of interaction among various stakeholders, as they allow project members to relate heterogeneous perspectives. In all three examples presented in this section, the visualisations supported the communication among various members of the organisation and helped them communicate their respective perspectives, which were rooted in their specific work practices. They became collaborative artefacts. This is especially important in action research, where the researcher, together with software engineers, users and other stakeholders, must understand needs and deliberate change. For this purpose, it is critical to share knowledge among individuals with different backgrounds, perspectives and motivations. Visualisations promote the verbalisation of implicit understandings as words and phrases and the collaborative creation of new concepts through dialogue.

To promote dialogue among researchers, software developers and other stakeholders, the form of notation used must be well chosen. Software engineers and researchers are familiar with formal state machines and semi-formal notation, such as UML diagrams. When they use such notation, the focus is often on using the notation correctly, and the correct mapping of what should be depicted onto the model. Therefore, these forms of notation often limit the discussion to the correctness of the model. Though formal notation may play an important role, as in the comparison of the software products in the SIM case, we often use more flexible, expansive [19, 40] forms of visualisation, such as the rich pictures introduced in the toolkit part of this section. Such deliberately informal notations allow the addition

of new perspectives to the picture and provide space for opposing views of the same reality.

Visualisation Example: IU—Mapping of the IT Infrastructure
As part of the field work, the action researcher at IU created a visualisation—a map—of the organisation's IT infrastructure. The map was developed through meetings with IU's only (at the time) external IT developer.

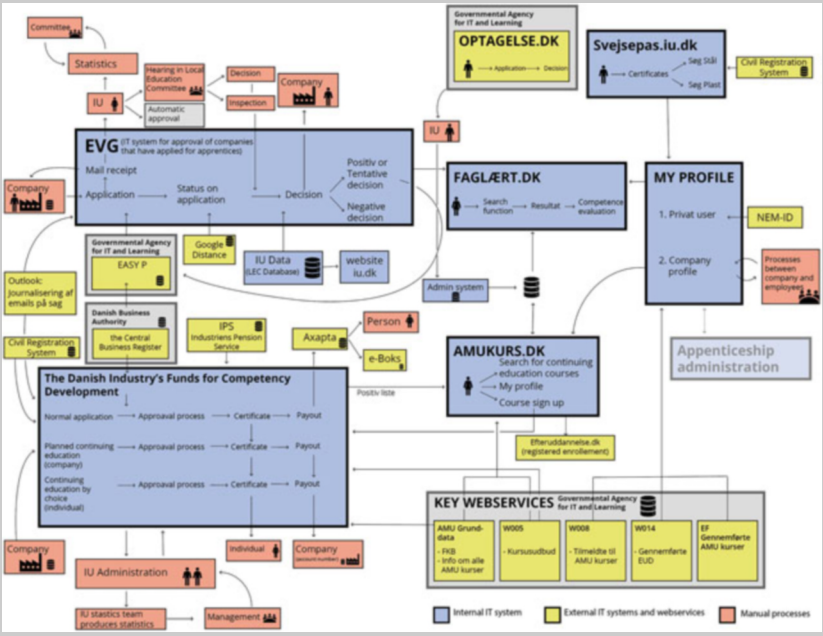


Fig. 7 Case 2: Mapping IU's IT infrastructure

The organisation had never developed a representation of their IT Infrastructure before; therefore, this visualisation (see Fig. 7) became a tool for explaining how the internal systems were integrated and how and to what extent the organisation's IT infrastructure—and by extension, data practices—depended on external stakeholders' IT systems and Web services. Over time, visualisation was used as part of the reasoning for establishing an internal IT department and in this way improve the internal understanding of IT and the extent to which the organisation's IT infrastructure—and, by extension, data practices—depended on external stakeholders' IT systems and Web services.

Finally, visualisations are a powerful tool for communicating research and research results. Figure 2 in the introduction of the IU example, which visualises the

action research process, is the final version of a series of figures. An early version was developed at the beginning of the project; it was revised and used throughout the project to discuss progress and changes to the plans. The final version demonstrates the action research that was carried out. When presenting research results, visualisations may similarly play an important role: an example is Fig. 10, which was first published in as part of a conference article [42]. The figure presents a diagram of the public sector arena for vocational education and training in Denmark. The diagram has been crucial to an understanding and the interdependence of data and cross-organisational collaboration and to communicating the findings to the academic community.

Visualisations may be used to communicate research findings to industrial practitioners and domain experts, both within the project and in presentations to external industrial practitioners. The downside of visualisations is that good ones take time to create. Often, they are not created on the spot but used as a tool to assist thought and accordingly are revised and developed over time. Also, not everybody is comfortable displaying his or her (lack of) drawing skills. However, we found in most cases, that the result is worth the effort.

Toolkit Visualisation

Visualise, Check with Relevant Collaborators, Repeat

When complex circumstances are visualised, heterogeneous perspectives often need to be reconciled. In these cases, iterations of visualisation, checking the representations with relevant members of the organisation and repeating are important: that allows the stakeholders involved to express and explain their perspective. In two of the examples in this section—the WMU example of establishing mutual understanding through rich pictures and the IU example of mapping the organisation's IT infrastructure—developing the visualisation was part of the research process. In the WMU example, the use of rich pictures was part of an intervention that demonstrated the scaling of participatory design methods at an organisational level. In the IU case, the visualisation was about mapping the data sources that were used to prepare and plan the intervention. For example, the map was used to identify the subject system of the first action research cycle.

Rich Pictures and Inspiration from Design Artefacts

It is challenging to create good and relevant visualisations, because they often need to convey complex information in a relatively simple manner, in order for it to resonate with the people and the project to which it relates. To begin with, we suggest looking at existing visualisation methods and, if necessary, adapting them to your needs. One of our favourites is the rich picture, a deliberately informal notation that is designed to be adapted and extended to the needs of the specific design or research purpose [30]. The set of elements in Fig. 8 provide an inspiration for own adaptation. If new visualisation

(continued)

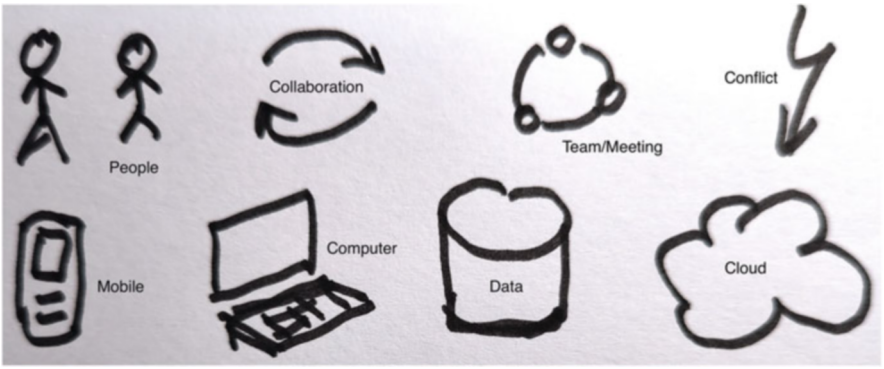


Fig. 8 Examples for Rich Picture symbols

elements are needed during a discussion with domain experts, they may be designed on the spot. Conflicting perspectives and unclear elements may be 'put on the table' or the whiteboard as crossed swords or clouds.

Many user-centred design methods offer a number of graphic representations that typically may be flexibly adjusted, such as service blueprints, ecosystem mapping or timeline mapping.

Exercise 4: Rich Pictures
Develop a Rich Picture of Stakeholders and Interests based on the Stakeholder analysis in Exercise 2. Explain the rich picture to a colleague, and enhance it based on his or her questions about the case. Add the rich picture and a description of it to the stakeholder analysis and the project description.

5 Challenges as Learning Opportunities

The core advantage, and also the main difficulty of participatory action research, in particular, is that it offers the opportunity for industrial practices and the setting in which software is developed to 'talk back' to the methods, techniques and tools proposed by the researcher: e.g. methods proposed by researchers may be controversially discussed by the software developers involved, or an intervention decided on does not happen for some reason. The challenges that emerge during the intervention often indicate blind spots in the established research. One example

of this is the SIM case, below: Traditional software architecture research focusses on software architecture design and communicating the software architecture to the developers. When long-living software products are developed, communication about the difficulties faced by the existing architecture and the reasons why work-arounds occur are equally important. However, these needs tend to be neglected in the scientific discussion.

Challenges Example: SIM—Being Challenged to Investigate the Concept of 'Walking Architecture'

One of the main surprises for the collaborating software engineering researchers was the absence of documentation of the software architecture not only of the software product that was subject to the collaboration but to all software products maintained in the company. There technical leads would teach new developers as apprentices. They also doubted the usefulness of a document that described the structure, as it would soon be outdated. The researchers wondered whether the distrust of software architecture documentation was due to the specificities of software product evolution or whether it might be due to the developers being mainly applied mathematicians. To find out more, the researcher held a set of interviews with lead architects, tech leads and core developers of software products, from SMEs to product teams in multinational companies. The interview study [50] confirmed that most of the product teams interviewed did not document their software architecture. The comparison with the only (open-source) software product that used a wiki to capture the software architecture indicated the rationale behind the lack of software architecture descriptions in other cases: The interviewee explained that every evening, he needed to read the commit statements and review the changes to the wiki so he could intervene if a change was not in line with the rationale behind the design architecture. Without the extra effort, he would be unable to keep up to date. In other words, there might be good reasons for bad software architecture documentation that makes it necessary for the developers to consult the architect before changing the structure of the software. In the final part of the field work, the team supported the action researcher in describing the core features of the architecture after it was re-engineered to support the simulation departments, if project-specific features needed to be included. (Such customisations were needed rather frequently and provided one line of input for the evolution of the software: if a project-specific feature was deemed relevant to other projects, it would often be revised and included in those products.) Here, there was no danger that the existence of documentation could result in an accidental evolution of the software architecture of the product.

In other cases, the challenges faced during deliberation and intervention indicate connections between subjects that are normally treated in different research communities. In Sect. 3, we shared an example in which the deliberation resulted in a method welcomed by the interaction designers, software developers and technical management involved in the development that could not be applied in the organisation, because of the way the company collaborated with its customers. Here, the business model affected the applicability of an interaction design method. Such dependencies are neither subject to Interaction Design and Participatory Design, nor are they normally discussed in the economics literature. Similarly, in the WMU case, the example below describes how the action research evolved from focussing on co-design methods in organisational settings to the innovation of the technical base and the implementation of participatory IT Management structures. Again, the results indicate a connection between Co-Design and Information Systems that is normally not addressed in any of the related discourses.

Challenges Example: WMU—From Co-design Methods to Redesign of the Technical Infrastructure and Participatory IT Management

IT development at the WMU was characterised by a strong emphasis on the expertise and support of the faculty and administration. The research project was launched to strengthen and further develop this Co-design tradition. The first action research cycle was implemented as part of the redevelopment of the teaching support, the scheduler of courses and rooms. One of the results showed that in many cases, implementing the interaction required by the domain experts would be impossible, given a university's IT budget. At the same time, relevant parts of the functionality were implemented with standard software. As a result, the next action research cycle addressed the development of the technical basis of the IT infrastructure to allow for the use of standard office systems to edit data, such as Microsoft Excel™ for editing course and room schedules. Such a change in the technical infrastructure needed to be deliberated and determined by a computer committee that consisted of representatives from the IT department, various administrative departments and faculty. This led to exploring ways of making such complex changes that only very indirectly affect users comprehensible to members of the organisation who were not IT professionals. At the same time, the WMU renewed its commitment to a co-determination approach to IT infrastructure development with the above-mentioned computer committee. During the process that led to the confirmation of a participatory IT Management strategy, it became apparent that in order to effectively steer an increasingly complex IT infrastructure, better tools of IT Management were needed. These means were then subject to a third-action research cycle. To answer this need, the topic of the PhD study developed from participatory design and end-user development to sustaining participatory design in an organisation [8].

Though the examples indicate action research's potential for identifying and addressing blind spots in and between various research discourses, often these developments are difficult to handle. And there is no one way to address such challenges. In the SIM case, the result was a side project that supported and triangulated the findings of the main study. In the SIM case, the insights shared by the practitioners and those based on the interview study informed the design of the intervention. In the other two cases, the difficulties encountered resulted in a change in the scope of the PhD dissertation. At the WMU, the second-action research cycle focussed on the development of an infrastructure that allowed for the integration of standard systems for specific tasks. In the case in which Personas were not applicable, the subsequent research investigated how the communication between interaction designers and software developers improved [36]. As the results questioned the role of methods, the reflection on methods and their roles was part of the final dissertation [34]. In any case, both the PhD researcher and the supervisor needed to be flexible, to be able to respond to difficulties encountered and to turn them into learning and research opportunities. However, this may be a challenge if a dissertation is involved. Students often have difficulties to estimate the complexity of a doctoral project. The supervisor has an important role in guiding the student, both academically and with respect to the practical aspects. For example, students are easily overwhelmed by the opportunity to make improvements or underestimate the importance of navigating the organisational setting, which may present conflicting stakeholder interests. It may not be possible to determine the scope of the research before starting the empirical research; action research cycles may be an ongoing process, as the researcher and practitioner gain insights (through the interventions) into the work domain. Therefore, it is vital that the PhD student and the supervisor meet regularly, to handle the evolving action research in a manner that is compatible with the requirements for the degree.

One of the tools that we used to manage the unfolding action research was debriefing, implemented in the research groups. Debriefings are an established part of qualitative research and aim to formulate a first insight into the collected data, with the goal of preparing a systematic analysis and directing future data collection. Debriefings help identify challenges early and explore them further, through discussions or through additional research. That way, a more nuanced understanding of the challenge and the root cause of the challenge may be obtained and become the basis for research publications and further research.

Toolkit Challenges as Learning Opportunities

Debriefing

Peer debriefing has long been discussed as a means of improving the quality of qualitative research [15]. Debriefing is also used as an emergency response, for psychological stress management [27] and for knowledge-

(continued)

sharing and learning related to future similar situations [14]. In our practice, peer debriefing served both purposes: The debriefing took place primarily as part of supervisory meetings between the supervisor (as internal debriefer) and the researcher. However, in the IU case, the debriefing also took place between the researcher and a member of the organisation. These debriefings would often focus on recent research activities and addressed organisational changes and/or development. The sessions were discussions between the two parties who asked questions and reflected on new insights or things that could be changed or improved.

In the cases of supervisor/researcher debriefings, the debriefer would ask questions to support the researcher's reflection on these events. In many cases, the debriefing focussed on the research methods and would result in decisions on next steps. However, special debriefing meetings focussed on the analysis and the substance of the research. They were scheduled either because of specific article writing projects or because of topics that came up in previous debriefing meetings. During these meetings, the debriefer typically asked about specific themes in the empirical work and explored the way the theme emerged in the field material, various aspects of the theme, related themes and so on. Such a debriefing meeting would typically end with a plan for further analysis or even with plans for additional data-collection.

Debriefings help the researcher cope with challenging situations; support knowledge-sharing, especially as a starting point or part of a collaborative analysis; and allow the identification of the scientific contributions of challenging developments and encounters.

Exercise 5: Debriefing
Form groups of three in the classroom. Ask the students to spend 15 minutes writing down their reflections on their empirical research. The students could consider a challenging situation, a situation where they felt their role as an action researcher was difficult, a situation that felt out of the ordinary or the like. In the groups, one student will act as the debriefer, one as the researcher and the third person will act as an observer. Each will take turns, to ensure that they experience debriefing from different perspectives. The person acting as the researcher should spend 5 minutes sharing the situation in question, and then the debriefer should ask questions. The debriefer may draw inspiration from 'The 5 Whys technique' [44], which is an interactive interrogative method used to explore the root cause of a problem. The technique may support the students' understanding of the reason for a 'strange' situation or why their role was challenged at a certain point.

6 Interventions as a Stepping Stone for Creating Sustainable Change

In addition to exploring the applicability of existing methods and developing new ones, action research may also aim to create sustainable change for the organisation with which the researchers are working, respectively, the software development of that organisation. There are several reasons to aim for sustainable change. Continuous use is a more substantial proof for the usefulness of a method or tool than if the same only works under supervision of a researcher; another reason is to ensure that the organisation involved benefits from the resources invested in the research project.

So, what does sustainable change look like? In this section, we share empirical insights that illustrate how action research may generate sustainable change and how to identify change in complex organisational settings. The aim is to help you through the process of identifying what constitutes change in your research project.

Action research initiates organisational learning that may reach beyond the research project. Initiating change is at the core of the interventions that are part and parcel of action research. Furthermore, companies and organisations become involved in action research with the hope of improving their way of developing software. However, it may be very difficult to determine whether an intervention has effected sustainable change that persists beyond the end of the research collaboration. And, sometimes, the longer-lasting changes in an organisation are not directly related to the interventions of the action research.

Action research does not have to explicitly aim for long-term change for the collaborating organisation. Especially for MSc or PhD students, evaluating a new method or tool with a single development project may be the only realistic goal. Action research is a time-consuming undertaking, and typically, students are constrained with regard to the time that is available for them to work on the dissertation, including carrying out empirical research, analysing outcomes and reporting the results. At the same time, even a project that focusses on a single application development may have a valuable outcome, both academically and for the organisation where the research has been carried out. Nguyen's M.Sc. thesis at the World Maritime University is an example of a student carrying out one action research cycle, in this case, to design and test the usefulness of an educational technology solution for training seafarers during the COVID-19 pandemic [31]. The prototype that the student developed yielded insights into how online, blended-learning modalities could be used, where the seafarers were unable to meet physically for their training.

In many action research projects, the sustainable change in the organisation is both appreciated and supported. The interventions implemented together with the collaborating team focus in the first place on changing this team's specific development practices. The interventions often address both the tools and the use of technology and the team's work practices. These local interventions may change the configurations of software development tools and techniques used by

the organisation and the organisational frameworks for, and methods of, software development. The research is geared to adding to the body of knowledge of software engineering and eventually influencing the standard tools, methods and processes available to the organisation. We have formulated these levels of sustainable change as a framework in the toolkit box below.

Toolkit Sustainable Change

Understanding Sustainable Change

The framework below structures the understanding of how action research learning and interventions for sustainable change take place in an organisational setting. The framework in Fig. 9 is inspired by an article on the impact of participatory development of educational technology on organisational processes [9] and builds on the work of Bødker et al. [3] and Pipek and Wulf [33].

The horizontal layers indicate the need for participatory knowledge development processes related to domain-specific knowledge (top horizontal layer) and technical knowledge (bottom horizontal layer), which, when combined, result in knowledge development related to the use of technologies (middle horizontal layer). This knowledge development, in turn, relates to the state of the art regarding techniques and methods (left-hand column); organisational repertoires of methods, processes, tools and techniques (middle column); and in situ development and learning in a specific project (right-hand column).

Action research often starts at the individual project level. It may focus on either the appropriation and innovation of techniques and tools and the appropriation and innovation of methods and processes, and in many cases, it addresses how both the social and the technical aspects interact. As the deliberation example on page 3 indicate, the organisational history and structure influence the specific development practice and the outcome of an intervention. Action research may explicitly address the organisational layer, for example, by deploying innovative tools or introducing methods to a wider part of the organisation. Even if the action research does not aim for innovation at the organisational level, it may influence the organisational level secondarily. The organisational level is informed by state of the art of software engineering methods and techniques communicated through professional networks or media or through commercial and open-source tools. This state of the art also informs the action research through the researcher. Eventually, the research results will feed back into the body of knowledge.

The learning processes indicated in the framework by the looping arrows build on Susanne Bødker et al.'s concept of 'knotworking' [4]. They show how 'backstage' development activities in the organisation are crucial to the sustainability of changes initiated by a specific action research project 'onstage'.

(continued)

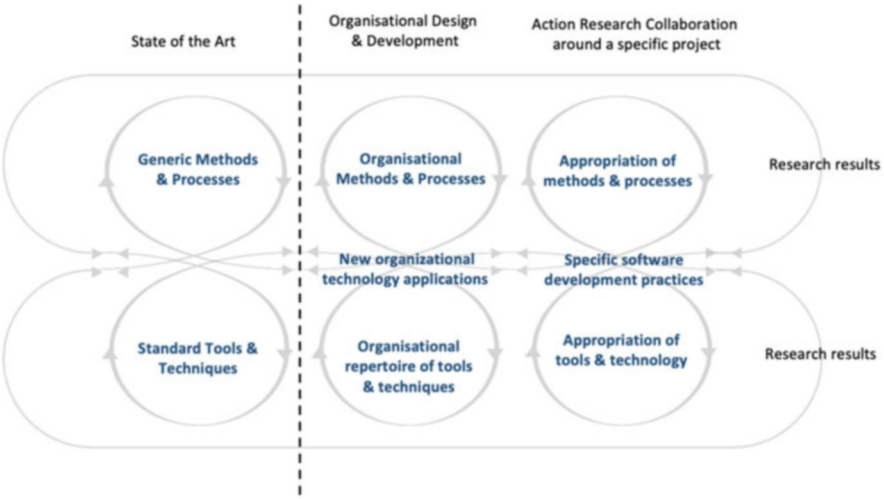


Fig. 9 Framework for organisational analysis of action research Intervention

Given this understanding, action research focusses on the application and development of knowledge, both in the specific development setting and in connection to the learning processes across the various levels.

All cases presented in this chapter serve as examples for creating sustainable change and innovation, both as intended parts of the research and in the form of secondary effects. In the SIM project, the focus was on introducing Agile software architecture practices that supported the continuous evolution of a software product. The introduction of software architecture concepts, especially the concept of product line architecture and the results of an exploratory comparison of the code bases of two sister products, had ripple effects on the organisation, beyond the specific context of the project. For example, it supported—if it did not prompt—the decision to integrate the two related products. The project continued to develop lightweight methods and tools for maintaining the architecture of continuously evolving software products, based on the study and intervention of the re-engineering project for the one-dimensional simulation engine. The example from the SIM project, below, highlights how changes to the development infrastructure may support the sustainability of organisational interventions.

Sustainable Change Example: SIM—Using Artefacts to Support Organisational Changes (Build Hierarchy)
The sustainability of the introduction of new methods and processes may be supported by materialising changes in the development infrastructure: In the SIM case, the researcher configured the build hierarchy, so violations of the software architecture's dependency constraints were discovered when the executable for testing was built [46]. The team, especially the developers who joined the team, regarded this as support. They felt that the additional structural check served as a safety net that, together with the product's new overarching architecture, allowed them to become productive much more quickly than before: historically, the company estimated that several months up to a year and a half were needed for a new developer to understand the hydraulic simulation software well enough to contribute to its development; with the new architecture and the support through the built hierarchy, the time was reduced to a few weeks.

The configuration of the build system allowed the architecture to become a support for day-to-day development and at the same time established affordances for discussing the architecture when needed. It also ensured that architecture evolution was implemented explicitly, as it required changes to the configuration of the development environment.

In the WMU case, the PhD researcher began by exploring participatory design methods for the development of one specific application. The experience of the concrete project led to a change in the technical framework used to develop teaching and learning support at WMU, which was trialled in a second development project. In parallel with this, it became apparent that the IT-management structure and tools needed to be reconsidered to handle such substantial changes to the IT infrastructure in an informed and accountable manner, which resulted in methodological and organisational interventions at that level. The new IT project-management approach was implemented and evaluated in a third development project. The example below illustrates how these dynamics unfolded.

Sustainable Change Example: WMU—Sustainable Change as Collaborative Organisational Learning
The way in which the action research cycles in the WMU case alternated between the ICT system and method improvements at the local-level project with organisational-level improvements promoted sustainable change. The action research was anchored in the local level in the development of a Learning Management System, a Student Life-Cycle Management system and

(continued)

a Contact Database System that were used in everyday work by employees of the WMU. The action researcher collaborated closely with the associated domain experts. The software development that he contributed was validated in use. The local development iterations led to more substantial organisational level changes. One example of the connections established is illustrated by an early problem in the first action research cycle, when the researcher developed a new Web-based scheduling system to replace an Excel-based scheduler. The administrator working with scheduling did not find it useful: 'You were just not listening to me [...] that is not going to work'. The development needed to be redone.

The problems encountered when developing the scheduler reoriented the action research and prompted two succeeding interventions at the organisational level: (a) The technical infrastructure that underpinned local software development was upgraded to support interfaces with office desktop systems. The upgrade also resulted in a revised architecture and a new programming language. (b) The upgrade, in turn, provided for new ways to develop software, as it enabled the rapid prototyping of new functionalities, which made it easier to test their usefulness with the domain experts. In this way, not only did the lessons learned from the scheduler application result in an improved version of the scheduler but also in infrastructure and methods improvements.

Anchoring the interventions in the users' and the organisation's actual needs, and evaluating technical and methodological interventions in new software development projects, resulted in collaborative learning that, in turn, resulted in a long-term and sustainable change in the WMU case.

In the IU case, sustaining the change was part of the action research cycles from the very beginning. The first cycle focussed on exploring methods for designing with data related to a specific project. The second intervention and cycle targeted the organisation and their capability to work with design and innovation projects. The action researchers adopted methods from service design and supported various smaller-scale change projects in the organisation, so-called service design micro-cases [43]. The final intervention and action research cycle focussed on supporting some members of IU to work more independently with the design of data [41].

Exercise 6: Reflecting Sustainable Change

Using the framework in Fig. 9, reflect on how your project, or an action research project you found in the literature, is positioned in relation to the framework. Think about the project's focus and its action research cycles and

(continued)

how those cycles relate to specific projects and the end user and organisational infrastructures and development methods. Furthermore, consider the project participants' learning requirements and the support that participatory tools and techniques may offer them.

7 From Action to the Research Results

Most action researchers might confirm that interaction with industrial practices is an adventure. However, the purpose of the research is the development of scientifically accountable results that may form the basis for future research, be it in the form of new action research or more controlled experiments. Publishing action research is challenging:

1. As action research often relies heavily on qualitative empirical research, it presents the challenges that come with such flexible approaches: the research and analysis need to assure the trustworthiness of the research. Furthermore, as with all flexible research approaches, the research itself may result in changes to both the interventions and the empirical methods applied.
2. Applying action research entails a conscious intervention in the observed practices. This entire intervention process needs to be regarded as data, which means all interactions with the team, including the analysis of the situation, deliberation on interventions, implementation of the interventions and evaluation are part of the research and need to be documented and becomes empirical data.
3. To document the efficiency and effectiveness of an introduced method, additional research may be needed to establish a baseline before the intervention, to make it possible to compare this to the situation after the intervention. This baseline may be quantitative or qualitative. For example, if the agreed-on intervention is meant to improve the test efficiency of the development, the current test efficiency may be measured using quantitative measures. If the intervention should improve the communication between interaction designers and software engineers, the baseline may use qualitative interviews and the documentation of today's development and communication practices as a baseline. The evaluation of the intervention needs to include corresponding methods. Often, the same quantitative measures may be applied to the new situation; using qualitative research, it may not make sense to ask the same questions as were asked previously; however, the same developers could probably be interviewed again when inquiring into the same aspects of the development.
4. Software development is complex: smaller and larger groups of developers work in parallel; developers communicate with product managers, users and customers where appropriate; sometimes, the development is distributed over several sites or is even fully dispersed, with only virtual collaboration. The object of work,

the software, is defined by program code and based on and coordinated through a number of documents, which today reside in development environments. Aspects relevant to the research may not be readily apparent to the researcher. This means that the core action research may need to be complemented by suitable additional methods, such as source code and document analysis, or interviews with individual developers or entire groups.

Newcomers and students who implement action research for the first time are often overwhelmed from the sheer amount of data. We recommend starting with the analysis and writing early in the process, to identify relevant research themes that underpin and are relevant to the research focus from which the action research started.

Analysing the initially collected data and relating the findings to the relevant literature supports a reflection process that may also inform the deliberation and interventions. Also, later in the research, individual parts of the action research may be published as stepping stones. 'From Action to Research Results Example: IU—Using publications as stepping stones during reflection' demonstrates how writing articles may become part of the reflection process.

From Action to Research Results Example: IU—Using Publications as Stepping Stones in the Reflective Process

The PhD dissertation based on the IU case resulted in six published papers (and two additional papers which were part of a related study). The papers were published during the course of the PhD programme, which was completed in 3.5 years. This required ongoing sequences of planning, conducting and analysing the research and writing about research findings. The papers built on each other. For example, the first paper focussed on understanding the current data handling in the organisation [39]. The analysis of the initial research presented two important research points: (a) that IU could be considered a 'knowledge broker', and that of cross-organisational collaboration needs to be considered in the action research, and (b) the concept of human-data interaction. The publication was a stepping stone for reflection, as it helped us understand that if we were to consider design with and of data in the IU setting, we could not consider the organisation in isolation. We also had to consider the organisation's key stakeholders and collaborators. This informed the first intervention and influenced the design of the subsequent action research interventions. Over time, we developed the diagram in Fig. 10 that depicted the complex network in which IU navigates and gives insight into the many stakeholders that need to be considered in relation to data handling [42].

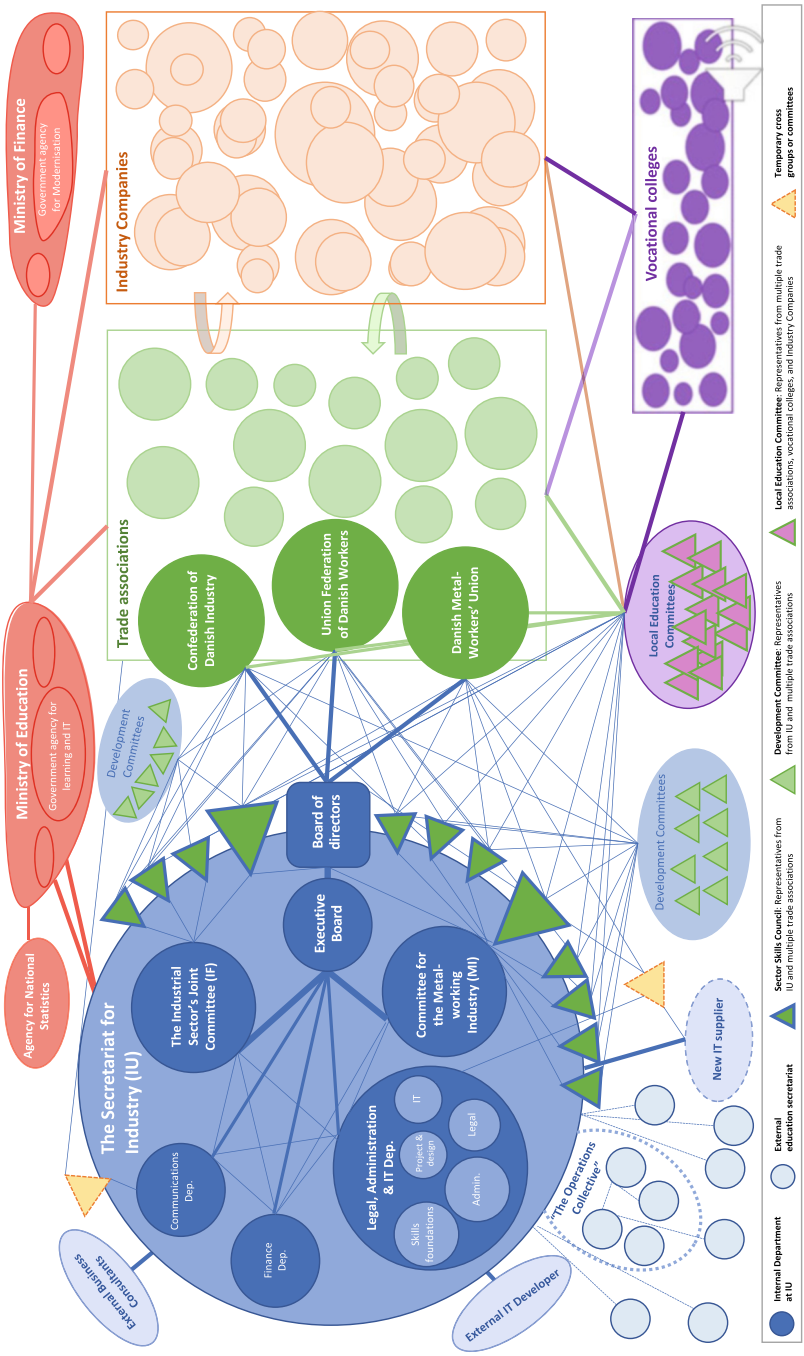


Fig. 10 Case 2: Diagram of the public sector arena for vocational education and training in Denmark [42]

The themes of such early publications are often suggested by debriefing meetings with the research group or supervisor (See Sect. 4, Challenges as Learning Opportunities). Such debriefings also help identify so-called ethnographic rich points (Agar [1]), episodes encountered by the researchers that are experienced as unusual or surprising, and often lead to a deeper understanding of the research subject. In ethnographic and anthropological research, such rich encounters often provide a starting point for the analysis of the field material. These meetings then need to be followed by a systematic targeted analysis, for instance, the interaction analysis of the data design workshops in the IU example above. The toolkit introduces a thematic analysis process and provides additional information about how to support data collection and analysis by looking for rich points.

The article, 'Organisation matters: how the organisation of software development influences the development of product line architecture' [49], is an example of how debriefing can be used to support the identification of relevant research results: debriefing led to the understanding that not only do technical requirements and design impact the evolvability of the software under discussion, but the software-development organisation, practices and business and use context matter as well. The results helped shape both the intervention and the developing research [46, 47].

When further exploring the results of a debriefing and/or ethnographic rich points, the research team may apply other suitable analytical methods. For example, when the team recognised that domain experts' ability to design with data was apparent in specific interactions during the design workshops, interaction analysis [25] was used to find, analyse and present examples of these exchanges [40]. Experience with a variety of social science methods is an advantage. In other cases, as in the WMU example below, a theme emerged that had not yet appeared in the core of the field work but had appeared as a border case at the fringes of the main research. In such situations, targeted empirical research, such as interviews or a focussed retrospective, may be designed, to triangulate with the existing field work. A second example is the above-mentioned article inspired by the SIM case, 'Software architecture awareness in long-term software product evolution' [50].

From Action to Research Results Example: WMU—Implementing Additional Research to Triangulate Field Work

The WMU case shows how triangulating the field work became important as the researcher was closely involved in the organisation. On the one hand, a close relationship between the researcher and the organisation provides an opportunity to acquire an insider's understanding of the development dynamics and to contribute to making useful improvements as part of the action research; on the other hand, a common concern that was accentuated in the WMU case is that the researcher is 'going native' and does not have enough reflective space or distance from the organisation to analyse the events

(continued)

taking place. Therefore, it became important that the supervisor supported the researcher during the field work. The contact database and electronic forms were two of the systems that were part of several action research cycles at WMU. An administrative assistant managed this development. As an end-user developer, she carried out the technical development herself. The role of the researcher was to support her by making improvements to the underlying technical infrastructure. To understand her work and deliberate improvements, the researcher used participatory observation and participatory design techniques.

When collecting data about end-user development practices, this became a problem: important aspects of her practices did not become explicit, as the administrator expected the researcher to know these aspects. To address these challenges, the supervisor interviewed the administrative assistant separately. In this way, several means of documenting the field work were employed, which triangulated each other and could be used for the subsequent analysis and report on the research.

Very few reports of full action research cycles have been published in core software engineering journals and conferences. One example from our research may be found in the article, 'Organisational IT managed from the shop floor: Developing participatory design on the organisational arena' [8]. Reporting action research in articles is often challenging, as the complexity of the research may not be easily condensed into 10,000 or even 20,000 words. Therefore, especially when PhD students are part of the research, the full account is often presented as a dissertation [6, 38, 47]. In their article, 'Style Composition in Action Research Publication', Mathiassen et al. provide a systematic literature analysis of action research publications in the Information Systems field, with respect to how the arguments and the contributions are presented [30]. They use the scheme adapted in Table 1 to categorise the articles they found.

In many action research projects, researchers and practitioners collaborate closely, sometimes so closely that the practitioners are part of the research development. If the developers involved are willing and able to put in the additional effort, they may even co-author some of the articles. As in most cases the practitioners involved do not have research skills, the team's researchers will still take the lead. To warrant co-authorship according to academic standards, all co-authors must have been involved in the reported research and should have participated in drafting and writing the article. The former would often present no problem in action research. To invite industrial software developers into the writing process, the researchers need to at least discuss outline and argumentation with the practitioners, provide the opportunity for the practitioners to contribute to the writing and allow time for the practitioners to review and add to at least one complete version of the draft. This

Table 1 Style composition in action research publications (Adapted from Mathiassen et al. [30, p. 351])

Premisses style	Inference style	Contribution style
Practical: Argument based primarily on challenges in software engineering practices **Theoretical:** Argument primarily based on challenges in software engineering research and theory	**Inductive:** Argument primarily grounded in evidence from the software development practices that are subsequently related to concepts from the research context **Deductive:** Argument primarily grounded in concepts from the research context, which are subsequently validated or illustrated by evidence from the software engineering practices	**Experience report:** Argument for insights from software development practices that may lead to research contributions **Field study:** Argument for a primary contribution to improve the knowledge about certain kinds of development practices **Theoretical development:** Argument for a primary contribution to a more general theory about software development **Problem-solving method:** Argument for a primary contribution concerning how to address certain kind of software development problems **Research method:** Argument for primary contribution to extend the knowledge of research methods and their application

may require more discipline with respect to time commitments during the writing process than usual in many academic contexts. Examples of action research articles co-authored by practitioners include [35, 36].

Toolkit from Action to the Research Results

Thematic Analysis

Thematic analysis is a common analytical method in the social sciences and may be used during both the problem analysis and the evaluation of the results in action research. Typically, a thematic analysis starts with identifying topics or coding recorded empirical material (e.g. from interviews, focus groups or workshops) and then proceeds with organising the empirical material into themes. This provides a basis for presenting the findings as a quotation-rich narrative. The coding may use a predefined scheme based on the research objectives and questions or be based on specific in situ insights (or a combination of both). For a more in-depth discussion, see the chapter on qualitative analysis by Treude in this volume. The example in Fig. 11 from the WMU case shows how codes with annotations are clustered into themes. At a basic level, simple tools such as sticky notes may be used to code the material, but qualitative data analysis software is useful for systemising the analysis, especially if the material is comprehensive.

(continued)

Ethnographic Rich Points
One approach to practically guiding the data collection and analysis is to look for rich points. Rich points are those surprises and insights that the researchers (and the participants) encounter during the action research [1]. They fuel ethnographic research, which is recommended for both the first needs analysis phase and final evaluation phase of the CMD approach. These may inform the coding in two ways.

- Encyclopaedic shared knowledge ethnography: What may be thought of as classic, encyclopaedic, 'shared knowledge' ethnography is referred to by Agar as 'disk contribution' [1, p. 12–16]. The goal is to find common threads, in the sense of 'patterns' or 'value configurations' that appear in several cases where the local and individual complications move to the background.
- Narrative ethnography: Narrative ethnography may be referred to as a 'pick contribution' [ibid]. It concerns the practices of everyday life, the way those practices are built on shared knowledge and all the other things that are relevant to the moment. It foregrounds rich points and their connection across domains and levels, rather than aiming to translate them into a generalisable pattern [2, p. 9]. This provides the capacity to identify individual 'complications and contradictions, not as evidence for the encyclopaedia, but as problems to explain in their own right' [1].

According to Agar's propositions, the foregoing two options are not mutually exclusive; instead, they can be and are often used together. A narrative account that keeps close to the details of an everyday development is combined with seeking common patterns in the empirical material. The latter may serve as the foreground or background to the former, but the idea is not to keep individual complications in the foreground while seeking common patterns among cases.

Exercise 7: Prepare for Publication
Discuss the Premise Style, the Inference Style and the Contribution Style (Table 1; see also Mathiassen et al.'s article [30]) for the action research project you have been developing in some of the exercises, and append the discussion and result to the project description.

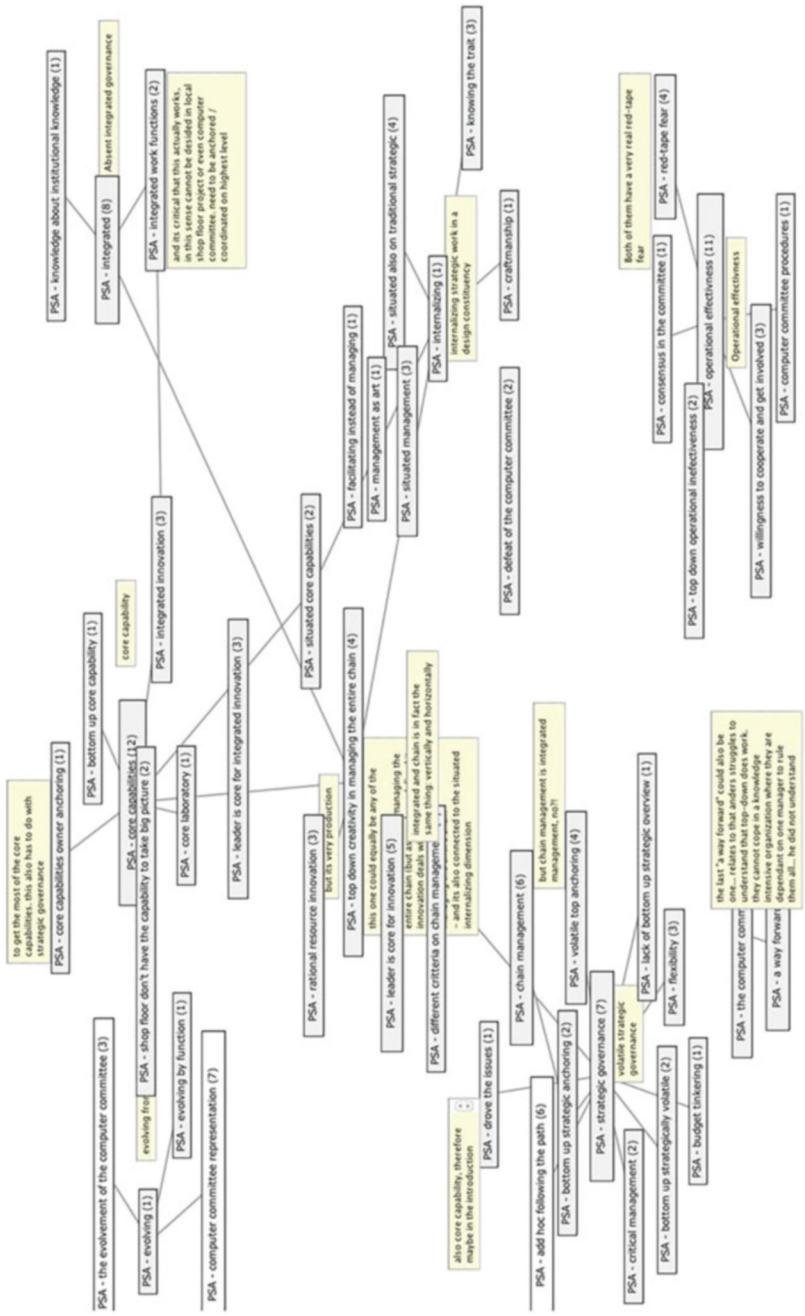


Fig. 11 Case 3: Example of codes structured into themes from the WMU case

8 Conclusion

Action research projects rarely play out exactly as originally envisioned. This chapter demonstrates how the challenging uncertainty of action research may be mitigated to allow the valuable research process and insights to be completed. Action research allows us to explore the appropriation and innovation of tools, methods and processes in real-world software development settings. It broadens our understanding of the factors that influence the adaptability of methods and helps identify how to improve the methods and to address new challenges—such as the co-design of data in the IU case and the interaction between software developers, domain experts and end-user developers at the World Maritime University—and the adaptation of existing methods to fit specific kinds of settings: the SIM case yielded a better understanding of how software architecture methods need to be fitted into software product development and evolution. Action research helped us extend the research beyond an understanding of software development environments through ethnography or grounded theory, for example, and it allowed to develop methods in situ, instead of ideating them and trying them out with the help of student projects.

Though we continue to apply action research, we cannot claim that it is unproblematic. In this chapter, we aimed to address some of its difficult aspects: the interactions with an organisation that is developing itself independently of the research; the participating organisation's reactions to the proposed intervention, which might result in changes to the research plans and might even lead to a change in the research question; and last but not least the publication of action research results. We also share tools, techniques and processes that we developed or adopted from others' work, and that helped us meet the challenges of action research. We hope that after reading this chapter, the readers will be inspired and understand some of the concerns of being involved in action research and how to address them.

Acknowledgments Writing this chapter provided us with the opportunity to reflect on the didactic aspect of our teaching and learning about action research. In particular, it enabled us to collect and reflect on the tools we use to address the sometimes-challenging interactions with industrial software and IT development. We would like to thank the editors for providing us with this challenge. The reviewers of this chapter helped us develop our conviction that this unusual piece would eventually fall into place. Last but not least, we would like to thank the collaborators in the various projects who contributed to our learning and supported the further development of our understanding and practice of action research.

References

1. Agar, M.H.: The Professional Stranger: An Informal Introduction to Ethnography. Academic, San Diego (1996). papers3://publication/uuid/96F108B0-0BDC-4D1B-AAF2-BB963B9F08E9. http://www.citeulike.org/group/268/article/221525
2. Agar, M.: Culture: can you take it anywhere? Int. J. Qualit. Methods **5**(2), 1 (2008). papers3://publication/uuid/07429980-CBE1-45C1-8E90-B6E59B9529CA. http://www.ethknoworks.com/

3. Bødker, K., Kensing, F., Simonsen, J.: Participatory IT Design: Designing for Business and Workplace Realities. MIT Press, London (2004)
4. Bødker, S., Dindler, C., Iversen, O.S.: Tying knots: Participatory infrastructuring at work. Comput. Support. Cooperat. Work **26**(1), 245–273 (2017). https://doi.org/10.1007/s10606-017-9268-y
5. Beck Kentand Andres, C.: Extreme Programming Explained: Embrace Change. Addison-Wesley Professional, Boston (2000)
6. Bolmsten, J.: Sustaining participatory design in the organization: Infrastructuring with participatory design. PhD Thesis (2016)
7. Bolmsten, J., Dittrich, Y.: Infrastructuring when you don't - end-user development and organizational infrastructure. In: Costabile, M.F., Dittrich, Y., Fischer, G., Piccinno, A. (eds.) End-User Development: Third International Symposium, IS-EUD, vol. 6654, pp. 139–154. Springer, Berlin (2011). http://dblp.uni-trier.de/db/conf/iseud/iseud2011.html#BolmstenD11
8. Bolmsten, J., Dittrich, Y.: Organisational IT Managed from the Shop Floor: Developing Participatory Design on the Organisational Arena, pp. 383–417. Springer, London (2015)
9. Bolmsten, J., Manuel, M.E.: Sustainable participatory processes of education technology development. Educ. Technol. Res. Develop. **68**(5), 2705–2728 (2020). https://doi.org/10.1007/s11423-020-09803-3
10. Bryson, J.: Strategic Planning for Public and Nonprofit Organizations: A Guide to Strengthening and Sustaining Organizational Achievement. Jossey-Bass Publishers, Hoboken (1995). https://books.google.se/books?id=yydHAAAAMAAJ
11. Bryson, J.M.: What to do when stakeholders matter: stakeholder identification and analysis techniques. Public Manag. Rev. **6**(1), 21–53 (2004)
12. Checkland, P., Holwell, S.: Action research: its nature and validity. In: Information Systems Action Research: An Applied View of Emerging Concepts and Methods, pp. 3–17. Springer, New York (2007)
13. Ciborra, C.: From Control to Drift: The Dynamics of Corporate Information Infrastructures. Oxford University Press, Oxford (2000)
14. Conoscenti, E., Martucci, G., Piazza, M., Tuzzolino, F., Ragonese, B., Burgio, G., Arena, G., Blot, S., Luca, A., Arcadipane, A.: Post-crisis debriefing: A tool for improving quality in the medical emergency team system. Intens. Crit. Care Nurs. **63**, 102977 (2021)
15. Creswell, J.W., Miller, D.L.: Determining validity in qualitative inquiry. Theory Pract. **39**(3), 124–130 (2000)
16. Dittrich, Y.: Doing Empirical Research on Software Development: Finding a Path between Understanding, Intervention, and Method Development, pp. 243–262. The MIT Press, Cambridge (2002). https://doi.org/10.7551/mitpress/6308.003.0016
17. Dittrich, Y., Rönkkö, K., Eriksson, J., Hansson, C., Lindeberg, O.: Cooperative method development: Combining qualitative empirical research with method, technique and process improvement. Empir. Softw. Eng. **13**(3), 231–260 (2008)
18. Eden, C., Ackermann, F.: Making Strategy: The Journey of Strategic Management. SAGE Publications, London (1998). https://books.google.se/books?id=rveTQgAACAAJ
19. Engeström, Y.: Expansive visibilization of work: An activity-theoretical perspective. Comput. Supp. Cooper. Work **8**(1), 63–93 (1999)
20. Eriksson, J.: Supporting the cooperative design process of end-user tailoring. Doctoral Thesis (2008)
21. Freeman, R.: Strategic Management: A Stakeholder Approach. Pitman, Boston (1984)
22. Hayes, G.R.: Design, action, and practice: Three branches of the same tree. In: Socio-Informatics, pp. 303–318. Oxford Academic, Oxford (2018)
23. Iversen, J., Nielsen, P.A., Norbjerg, J.: Situated assessment of problems in software development. ACM SIGMIS Database Database Adv. Inf. Syst. **30**(2), 66–81 (1999)
24. Jones, R., Seidelin, C.F., Neang, A.B., Lee, C.P.: Lessons learned from a comparative study of long-term action research with community design of infrastructural systems. Proc. ACM Human-Comput. Interact. **7**(CSCW1), 1–35 (2023)

25. Jordan, B., Henderson, A.: Interaction analysis: Foundations and practice. J. Learn. Sci. **4**(1), 39–103 (1995)
26. Kyng, M.: Making representations work. Commun. ACM **38**(9), 46–55 (1995)
27. Magyar, J., Theophilos, T.: Debriefing critical incidents in the emergency department. Emerg. Med. Australas. **22**(6), 499–506 (2010)
28. Mathiassen, L.: Collaborative practice research. Inf. Technol. People **15**(4), 321–345 (2002)
29. Mathiassen, L., Munk-Madsen, A., Nielsen, P.A., Stage, J.: Object-Oriented Analysis & Design, vol. 25. Forlaget Marko, Aalborg (2000)
30. Mathiassen, L., Chiasson, M., Germonprez, M.: Style composition in action research publication. MIS Quart. **36**, 347–363 (2012)
31. Nguyen Hoang, V.: Improving e-learning in maritime education and training: action research in the Vietnam maritime context (2021)
32. Nicolini, D.: Practice Theory, Work, and Organization: An Introduction. OUP Oxford, Oxford (2012)
33. Pipek, V., Wulf, V.: Infrastructuring: towards an integrated perspective on the design and use of information technology. J. Assoc. Inf. Syst. **10**(5), 447–473 (2009)
34. Rönkkö, K.: Making methods work in software engineering. Thesis (2005)
35. Rönkkö, K., Hellman, M., Kilander, B., Dittrich, Y.: Personas is not applicable: local remedies interpreted in a wider context. In: Proceedings of the Eighth Conference on Participatory Design: Artful Integration: Interweaving Media, Materials and Practices-Volume 1, pp. 112–120 (2004)
36. Rönkkö, K., Hellman, M., Dittrich, Y.: Pd method and socio-political context of the development organization. In: Proceedings of the Tenth Anniversary Conference on Participatory Design 2008, pp. 71–80. Indiana University. http://dl.acm.org/citation.cfm?id=1795245
37. Robson, C., McCartan, K.: Real World Research: A Resource for Users of Social Research Methods in Applied Settings. Wiley, Hoboken (2016)
38. Seidelin, C.: Towards a co-design perspective on data: Foregrounding data in the design and innovation of data-based services. Thesis (2020)
39. Seidelin, C., Dittrich, Y., Grönvall, E.: Data work in a knowledge-broker organization: How cross-organizational data maintenance shapes human data interactions. In: British HCI Conference 2018, pp. 1–12. BCS Learning and Development, Swindon (2018)
40. Seidelin, C., Dittrich, Y., Grönvall, E.: Foregrounding data in co-design– an exploration of how data may become an object of design. Int. J. Human-Comput. Stud. **143**, 102505 (2020)
41. Seidelin, C., Dittrich, Y., Grönvall, E.: Co-designing data experiments: Domain experts' exploration and experimentation with self-selected data sources. In: Proceedings of the 11th Nordic Conference on Human-Computer Interaction: Shaping Experiences, Shaping Society, pp. 1–11 (2020)
42. Seidelin, C., Lee, C.P., Dittrich, Y.: Understanding data and cooperation in a public sector arena. In: Proceedings of 18th European Conference on Computer-Supported Cooperative Work, vol. 4. European Society for Socially Embedded Technologies (EUSSET) (2020)
43. Seidelin, C., Sivertsen, S.M., Dittrich, Y.: Designing an organisation's design culture: How appropriation of service design tools and methods cultivates sustainable design capabilities in SMEs. In: Y. Akama, L. Fennessy, S. Harrington, A. Farago (eds.) ServDes.2020 Tensions, Paradoxes and Plurality Conference Proceedings, 2-5th February 2021, Melbourne, pp. 11–31 (2020). https://servdes2020.org. ServDes2020 Tensions, Paradoxes and Plurality Conference Proceedings, 2-5th February 2021, Melbourne; Conference date: 02-02-2021 Through 05-02-2021
44. Serrat, O.: The Five Whys Technique, pp. 307–310. Springer Singapore, Singapore (2017). https://doi.org/10.1007/978-981-10-0983-9-32
45. Star, S.L., Griesemer, J.R.: Institutional ecology, translations' and boundary objects: amateurs and professionals in Berkeley's museum of vertebrate zoology, 1907-39. Soc. Stud. Sci. **19**(3), 387–420 (1989)

46. Unphon, H.: Making use of architecture throughout the software life cycle - how the build hierarchy can facilitate product line developments. In: Proceedings of the 2009 ICSE Workshop on Sharing and Reusing Architectural Knowledge, pp. 41–48. IEEE Computer Society Press, Washington (2009). https://doi.org/10.1109/SHARK.2009.5069114. International Conference on Software Engineering; Conference date: 02-07-2010
47. Unphon, H.: Architecture-Level Evolvability Assessment: Assessing Sustainability of Software Product Evolution (2010)
48. Unphon, H.: Re-engineering for evolvability: Considering social as well as technical requirements for software products. Ph.D. Thesis, Denmark (2010)
49. Unphon, H., Dittrich, Y.: Organisation matters: How the organisation of software development influences the development of product line architecture. In: Software Engineering, SE 2008, February 12–14, 2008 Innsbruck, pp. 178–183. ACTA Press, Calgary (2008). Er den peer-reviewed ? Ja.; Software Engineering, SE 2008, as part of the 26th IASTED International Multi-Conference on APPLIED INFORMATICS; Conference date: 12-02-2008 Through 14-02-2008
50. Unphon, H., Dittrich, Y.: Software architecture awareness in long-term software product evolution. J. Syst. Softw. **83**(11), 2211–2226 (2010)

Teaching Case Study Research

Stefan Wagner

Abstract Case study research is an important research method in software engineering. Yet, it isn't easy to teach in a classroom setting. This chapter shares a proven teaching approach, the most important content to teach, and a deeper dive into qualitative analysis.

1 Why Should We Teach Case Study Research?

Case study research is a research method that provides an in-depth examination of a specific phenomenon or situation within a real-world context. This methodology is instrumental in software engineering research as it allows for the exploration of complex issues and phenomena in a detailed and nuanced manner. One of the primary benefits of case study research in software engineering is its ability to provide rich and detailed insights into software systems' design, development, deployment, processes, and teams in software development. By examining one or a few specific cases or situations in depth, researchers can identify patterns, trends, and best practices that inform future research and practice. This is particularly useful in a field where best practices constantly evolve and improve.

To underscore the methodology's practical applications, consider the seminal work of Amershi et al. [1] on software engineering for machine learning. This case study delved into the workflow of developing AI-based applications, providing rich insights that directly influenced subsequent research and industry practices. Such examples demonstrate how case study research can yield actionable knowledge transcending the theoretical realm.

However, teaching case study research can be challenging due to its complexity and subjectivity. Unlike more quantitative methods, case study research relies heavily on qualitative data and interpretations, which can be difficult to standardize

S. Wagner (✉)
TUM School of Communication, Information and Technology, Technical University of Munich, Heilbronn, Germany
e-mail: stefan.wagner@tum.de

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2024

D. Mendez et al. (eds.), *Handbook on Teaching Empirical Software Engineering*,
https://doi.org/10.1007/978-3-031-71769-7_16

or replicate. Additionally, the subjective nature of case study research can lead to a wide range of interpretations and conclusions, depending on the researcher's perspective.

Furthermore, misusing the term "case study" in the software engineering literature further complicates teaching it. Unfortunately, for many researchers in software engineering, any example or proof-of-concept done on a concrete program or software system can be labeled a "case study." This dilutes the term and devalues case study research in general. In particular, it isn't apparent that students read papers that contain so-called case studies that do not resemble the methodology discussed in class.

Therefore, to teach case study research effectively, it is crucial to provide students with a clear understanding of the methodology and its limitations. This involves discussing the strengths and weaknesses of the approach, as well as how to design and conduct a successful case study research project. Instructors should also provide students with examples of successful case studies in software engineering to illustrate the application of the methodology and help students understand the types of questions that can be answered through this approach.

Through hands-on exercises and real-world examples, students can learn how to identify a suitable case study topic, gather and analyze data, and draw meaningful conclusions based on their findings. By providing students with the skills and knowledge necessary to conduct high-quality case study research, instructors can help them develop a deeper understanding of software engineering research methods and their applications in real-world settings.

This chapter will provide a teaching approach that can be used independently or integrated into a more extensive research or empirical methods course. It has successfully been used with students on the bachelor, master, and doctoral levels. The main difference for these levels is how much detail can and should be covered. There are no prerequisites besides a basic understanding of statistics and empiricism. Furthermore, we will discuss using case study research to introduce or deepen the understanding of qualitative data analysis. Depending on whether there is a dedicated module on qualitative analysis, this can be more detailed or just a further example. While teaching this methodology can be challenging, providing students with a clear understanding of its strengths and limitations and practical experience in conducting case studies can help them develop the skills and knowledge necessary to conduct high-quality research projects.

The chapter is structured as follows: First, we will describe the learning objectives we aim for with the teaching approach. Next, we will summarize the essence of case study research. In the next section, we will describe a teaching approach that gives a sequence of topics effective for use in a classroom. Afterward, we will briefly discuss qualitative analysis as an essential method for case study research. In the next section, we will give some examples of exercises for students, describe pitfalls and best practices, and conclude the chapter.

2 Learning Objectives

The general learning objectives are that students (1) *understand* what a case study is and what possibilities and limitations it has and (2) be able to *apply* case study research for their own studies and research questions. As mentioned, the level of detail expected of the learning outcome differs depending on the educational level of the students. While a clear understanding and the ability to apply it to a small and straightforward study is a success on the bachelor level, doctoral students should be able to reflect on the limitations in detail and apply it to complex situations with several, possibly embedded cases.

We can break down the first general objective into the following detailed objectives:

- *Understand* what differentiates case studies from examples and proofs of concept.
- *Understand* when to apply case study research.
- *Understand* the role of case study research in larger research programs (e.g., real-world study of a phenomenon established in experiments).
- *Understand* the main terms and concepts, such as "case" or "theoretical saturation."
- *Understand* the steps in conducting a case study.
- *Understand* the limitations of case study research.

For the second general objective, there are also more detailed learning objectives:

- *Apply* case study design to create suitable and valid study designs for a study idea.
- *Apply* ethics analysis and approval.
- *Apply* data collection preparation.
- *Apply* evidence collection.
- *Apply* quantitative and qualitative data analysis.
- *Apply* reporting of the case study and its results.

3 The Essence of Case Study Research

In the following, we will cover all the content central to teaching case study research. Not everything has to be discussed in depth, but a basic understanding is necessary for the students.

As Runeson et al. [4] point out, case studies can be conducted in both inductive and deductive ways, depending on the research question and data collection methods. In software engineering, case studies are often used to investigate practical problems or situations that cannot be replicated in a controlled laboratory setting. For instance, a case study may involve analyzing an existing software system,

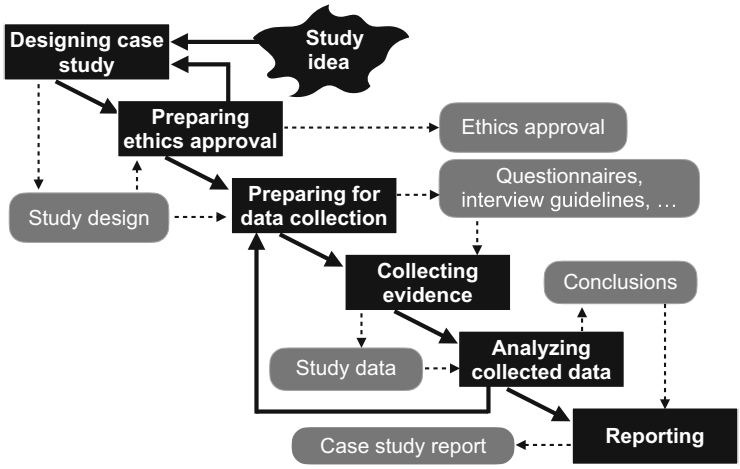


Fig. 1 The case study process. Darker boxes represent activities, and light-gray boxes represent artifacts. Solid arrows show the flow of time, while dashed arrows show the flow of artifacts

examining its development, and identifying potential issues or areas for improvement. This type of research can provide valuable insights into software systems' design, development, and maintenance, which cannot be obtained through most other methods.

The case study process typically involves several stages, including the case study design, ethics approval, preparation for data collection, collecting evidence, analysis of collected data, and reporting (see also Fig. 1). First, the researcher needs to establish a theoretical framework to guide the selection of cases and the formulation of research questions. This should be aided by conducting a literature review to find relevant theories, models, and best practices. Next, the researcher must identify and define the research question(s) or problem statement. Then, the researcher must choose an appropriate case study methodology based on the research question and available resources that align with the identified research question(s) and data collection methods.

The next step is getting ethics approval. For that, it is essential to fully understand the requirements established by the ethics board of one of the institutions involved in conducting the case study. This initial step is crucial to ensure complete compliance and to facilitate a seamless approval process. After thoroughly understanding the requirements, meticulously preparing the necessary documents based on the design outlined in the previous phase is imperative for submission to the ethics board. In some instances, some documents expected in the following phase of the project may also need to be submitted simultaneously. This proactive approach can effectively address any potential inquiries from the ethics board and significantly expedite the approval process. By being thorough and proactive, we can ensure a smooth and efficient journey toward obtaining ethics approval.

Data collection is a crucial stage in the case study process, as it involves gathering information from various sources, including interviews, observations, archival data, and quantitative data. Interviews are a standard data collection method in case studies, allowing researchers to gather rich and detailed qualitative data from participants. The choice of interview questions depends on the research question and the type of data sought; for example, open-ended questions may be used to gather detailed insights, while closed questions may be employed when specific information is required.

Observations are another important data collection method in case studies involving direct participation in the studied context. This can include observing user behavior, analyzing software code, or participating in meetings and discussions. Screen capturing and video recordings are also commonly used to collect data during observations.

Archival data, such as meeting minutes, test specifications, project plans, financial records, requirements specifications, source code, organizational charts, and reports, can provide valuable contextual information about the case study setting. Quantitative data, including sales figures, defect counts, timesheets, integration times and failures, and project costs, can also be analyzed to gain insights into the software development process.

Data analysis involves several stages, including coding, categorization, and theme development. Qualitative data is often analyzed using techniques such as content analysis or thematic analysis, while quantitative data may be analyzed using statistical methods. The choice of data analysis method depends on the nature of the data collected and the research question being investigated.

Finally, reporting and dissemination involve presenting the case study's findings clearly and concisely, following guidelines such as those outlined by Runeson and Höst [3]. Typical criticism of case study research includes generalizability, reliability, and validity concerns. However, when conducted carefully and with appropriate methods, case studies can provide valuable insights into complex software engineering problems.

While immensely valuable for an in-depth understanding of complex phenomena, case study research also has several inherent limitations that must be clearly explained to students to ensure the rigor and credibility of their studies. These limitations can broadly be categorized into issues related to subjectivity and bias, generalizability, reliability, and validity.

One of the primary limitations of case study research is the potential for subjectivity and bias. Because case studies often rely heavily on qualitative data, such as interviews, observations, and personal reflections, the researcher's perspective can significantly influence data collection and interpretation. This can lead to biased results if the researcher's expectations or beliefs affect how they gather and analyze the data. Participants may also introduce bias, consciously or unconsciously providing information that aligns with what they believe the researcher wants to hear.

Case studies typically focus on a single case or a small number of cases, which poses challenges for generalizing the findings to a broader population. Unlike large-

scale quantitative studies, case studies are context-specific and deeply rooted in the particularities of the studied case. This makes it challenging to apply the results to other contexts, as the unique conditions and variables of the case may not be present elsewhere. Therefore, while case studies provide detailed and nuanced insights, their applicability beyond the specific case studied is limited.

Reliability refers to the consistency of the research findings, meaning that the results should be replicable under similar conditions. In case study research, achieving high reliability can be challenging due to the qualitative nature of the data and the flexibility often inherent in the research design. Different researchers might collect and interpret the data differently, leading to variations in the findings. This lack of standardization in data collection and analysis methods can undermine the reliability of the study.

Validity concerns the accuracy and credibility of the research findings. Several types of validity can be threatened in case study research:

- Construct validity: Ensuring that the study accurately measures the concepts it intends to measure can be difficult. The abstract nature of many social phenomena and the reliance on qualitative data makes it challenging to develop precise and reliable measurement tools.
- Internal validity: This refers to how the findings accurately reflect the relationship between variables within the studied case. Confounding factors and the complexity of real-world settings can threaten internal validity, making it hard to draw clear causal inferences.
- External validity: Also known as generalizability, external validity concerns the extent to which the findings can be applied to other settings or groups. As discussed earlier, the context-specific nature of case studies limits their external validity.

Understanding these limitations is the first step toward mitigating them. Researchers can adopt several strategies to address these issues, such as triangulation (using multiple data sources and methods to confirm findings), maintaining a clear audit trail (documenting the research process in detail to enhance transparency and replicability), and seeking peer reviews and feedback to reduce bias. By acknowledging and addressing these limitations, researchers can strengthen the credibility and impact of their case study research.

Case study research, with its in-depth focus on individual cases or small groups, presents unique ethical challenges that researchers must carefully navigate to protect participants and ensure the integrity of the research. The close interaction between researchers and participants, the potential for deep personal insights, and the often-context-specific nature of the findings heighten the need for rigorous ethical considerations. Below are some fundamental ethical concerns that are important in general or specific to case study research.

Informed consent is a fundamental ethical requirement, ensuring that participants are fully aware of the nature of the study, their role, and any potential risks before agreeing to participate. In case study research, obtaining informed consent can be challenging due to the evolving nature of the study and the possibility of uncovering

sensitive information. Researchers must communicate the study's aims, procedures, potential risks, and benefits and ensure that participants understand their right to withdraw at any time without penalty.

Researchers conducting case studies may sometimes assume dual roles, such as being both a researcher and a practitioner (e.g., a software engineer doing research in their own company). These dual roles can lead to conflicts of interest and ethical dilemmas, as the researcher must balance their professional responsibilities with their research objectives. Transparency about these dual roles and carefully considering how they may impact the research and participants is essential.

The detailed nature of case study findings can pose ethical challenges in reporting and dissemination. Researchers must balance the need for rich, contextualized data with the obligation to protect participants' privacy and confidentiality. This may involve making difficult decisions about what information to include and how to present it in a way that maintains participants' anonymity. Additionally, researchers should consider the potential implications of their findings for the participants and the broader community, avoiding sensationalism or misrepresentation.

By rigorously addressing these ethical concerns, researchers can conduct case study research that is respectful, responsible, and ethically sound, ultimately enhancing the credibility and impact of their findings. In conclusion, case studies are a powerful research method in software engineering that offers a flexible and comprehensive approach to understanding practical problems and situations. By following a systematic process and using appropriate data collection and analysis methods, researchers can gather valuable insights into software systems' design, development, and maintenance.

A research method closely related to case study research is action research. It is a participatory and iterative method of inquiry involving researchers working collaboratively with participants to identify problems, implement solutions, and evaluate the outcomes concurrently. Unlike case study research, which primarily focuses on an in-depth analysis of a specific instance or phenomenon to understand broader principles or theories, action research is inherently interventionist. It seeks to create immediate improvements within the studied context. Yet both methodologies explore complex technical, social, and organizational issues. You find a more detailed discussion of teaching action research in Chap. 18.

4 Teaching Approach

This section will delve into a comprehensive teaching approach for case study research in software engineering. The goal is to equip students with the necessary skills and knowledge to undertake high-quality research projects.

Step 1: Start with an Example
To begin, provide your students with a real-world example of a software engineering problem that can be analyzed through a case study approach. This could be a recent

project that students are familiar with or one you have selected based on its relevance to the course material. The example should be detailed enough to allow students to understand the context and complexity of the problem but not so complex that it becomes overwhelming.

Step 2: Describe the Process
Once the students understand the general idea of the methodology, describe the process involved in conducting a case study research project in depth. This is the foundational framework for students to build their understanding of the methodology. This may include the following substeps as described above and shown in Fig. 1:

1. Based on the study idea, developing a case study design
2. Preparing ethics approval
3. Preparing for data collection
4. Collecting evidence through various sources such as interviews, surveys, observations, and existing data
5. Analyzing the collected data with qualitative and/or quantitative methods
6. Possibly going back to data collection preparation for the next round of data collection
7. Interpreting and drawing conclusions based on the findings
8. Reporting the findings

Step 3: Quantitative and Qualitative Analysis
Highlight the diverse spectrum of data analysis methods applicable to case study research. While emphasizing the importance of selecting the appropriate method based on the nature of the data and research questions, delve into quantitative methods (statistical analysis, data mining) and qualitative methods (content analysis, thematic analysis). This holistic approach prepares students to navigate the multifaceted data analysis landscape in software engineering case studies.

Step 4: Use Opportunity to Discuss How to Conduct and Analyze Interviews
Given the prevalence of interviews as a data collection method, provide students with a dedicated session on effective interview practices. Provide students with an opportunity to discuss how to conduct effective interviews, including the following:

1. Developing interview questions that align with the research question(s)
2. Identifying potential participants and obtaining informed consent
3. Conducting interviews using appropriate audio or video recording methods
4. Analyzing and coding transcripts for key themes and patterns

Step 5: Discuss Various Data Sources and Their Respective Challenges
Extend the discussion on data sources by detailing the challenges associated with each. Address data quality, validity, bias, and ethics issues for interviews, surveys, observations, and existing datasets. This critical examination of challenges enhances students' awareness of potential pitfalls in their future research endeavors.

Step 6: Difficulty in Simulating Whole Process, but Provide Raw Data to Analyze

While it may be difficult to simulate the entire case study research process in a classroom setting, you can provide students with raw data to analyze. This could include real-world datasets or fictional scenarios that align with the course material. Please encourage students to work in groups to analyze the data and draw conclusions based on their findings. If you use real interview data, ensure you respect the consent given for the data.

By following these steps, students will understand how to conduct a case study research project in software engineering. By using real-world examples and providing opportunities for hands-on practice, students will be better equipped to analyze complex problems and develop practical solutions in the field.

5 Qualitative Data Analysis

While quantitative analysis provides objective and often precise data and results, most case studies rely on qualitative analysis to gain a deeper understanding of the subject matter. Qualitative analysis is instrumental when dealing with small sample sizes or small amounts of quantitative data that limit the possibility of statistical analysis. To better comprehend the complexities of case study research, it is essential to delve into qualitative analysis.

Two primary qualitative data sources in case studies are interviews and existing documents. Interviews provide a wealth of information through open-ended questions, allowing researchers to gain insights into participants' thoughts, feelings, and experiences. Existing documents, such as texts, articles, and letters, offer a window into the past or current events, providing valuable context and background knowledge.

Qualitative content analysis is a versatile and widely used technique for analyzing textual data in various fields. Developed by Mayring [2], this method allows researchers to identify patterns, themes, and meanings in the data without being constrained by preconceived notions or theories. Researchers typically begin at the phrase or sentence level for analysis in social and environmental studies.

Researchers use various interpretation methods to analyze qualitative data, including summary, explication, and structuring. Summary involves condensing the material while still retaining its essential contents. Explication provides additional context on individual dubious text components (terms, sentences, etc.) to enhance understanding and explain or interpret the particular passage of text. Structuring involves filtering out specific aspects of the material, creating a cross-section according to pre-determined ordering criteria, or assessing the material based on predetermined standards.

By employing these methods, researchers can gain a deeper understanding of the data and identify patterns and themes that might have gone unnoticed with a surface-level analysis. The flexibility of qualitative content analysis makes it an invaluable

tool for exploring complex social and environmental issues. However, this method can be challenging for students to grasp as it does not follow a fixed routine like quantitative methods. Therefore, providing clear examples and guidance is crucial to help them understand the process.

One effective way to introduce qualitative analysis techniques to students is by using interview transcripts as a starting point. Teachers can give students these transcripts to analyze individually, allowing them to gain hands-on experience with qualitative analysis techniques. During class discussions, teachers can share their codings and interpretations of the data, emphasizing that there is no one "right" way to analyze qualitative data. This approach encourages students to think critically about the data and develop analytical skills. Additionally, teachers can provide feedback on the students' codings, fostering a collaborative learning environment.

Figure 2 shows an example of a made-up interview transcript that can be used in an exercise on qualitative content analysis. It shows the transcript of a hypothetical interview of a researcher with a developer about refactorings. We can now use summary, explication, and structuring methods in qualitative content analysis. In our experience, summarizing and structuring methods are the most useful in typical case studies. Table 1 shows an example table where a transcript of the observations of developers during static analysis is summarized and reduced stepwise. Table 2 structures similar data using predefined codes.

In the classroom, during an exercise in a lecture or a tutorial, we could let the students apply the summarizing method on the transcript in Fig. 2. The students can then compare their codes, and finally, the instructor can provide an example of codes and reductions.

Furthermore, a technique for evaluating the reliability of qualitative analysis needs to be covered. In qualitative content analysis, this is an inter-coder agreement. Researchers can give the text to a second person to analyze independently, compare their outcomes, and calculate inter-coder agreement using Cohen's kappa [5] or other methods. An alternative lightweight approach allows researchers to review the first coder's coding to assess the analysis's reliability. Both can also well be applied in a tutorial setting.

An essential concept in qualitative research is theoretical saturation, which occurs when additional data no longer contribute new insights to the research questions. This ensures that the analysis is thorough and that the findings are robust. Teaching students about theoretical saturation helps them understand when they have collected enough data to draw meaningful conclusions.

Finally, various tools are available to support qualitative data analysis, such as QDA Miner, Dedoose, and MaxQDA, which offer features like coding, categorization, and data visualization to facilitate the analysis process. By leveraging these tools, researchers can streamline their analysis and gain insights into complex social and environmental issues more efficiently. Yet it is sufficient to show these tools in general and run in-class coding sessions on paper. Introducing these tools to students without requiring their use helps maintain focus on understanding the fundamental principles of qualitative analysis without the overhead of learning new software. Demonstrating these tools provides an overview of the available resources, and

Interview Transcript of a Hypothetical Interview with a Developer about Refactorings
Q: Hello! Thanks for taking the time to answer some questions.
A: Hi! You are welcome!
Q: What do you know about the concept of "refactoring"?
A: Well, I have read about it and discussed it with colleagues over time. My understanding of refactoring is that we clean up our code after making functional changes.
Q: A standard definition is that code refactoring is the process of restructuring source code without changing its behaviour with the purpose to improve non-functional attributes of the software. What do you think about this definition?
A: Yeah. I think that fits quite well. I guess we sometimes do change some things that change the software behaviour during a refactoring.
Q: Why do you also change behavior?
A: If I realize during a refactoring that there is maybe a small bug or a little functionality that I can improve, I just do it.
Q: Please think about the last or current project you have been working on as a software developer. What refactorings have you used?
A: Let's see. . . I often use a rename refactoring. That's handy when I realize that I chose a bad name for a method or a class. Then, I regularly extract methods to create smaller, more understandable methods. Sometimes, I also move methods in the hierarchy of the inheritance hierarchy. That's also a refactoring, right?
Q: Yes, that's called Pull Up or Push Down.
A: Right. I think that's it.
Q: Okay. I have some more refactorings. Have you used those: Extract Class, Encapsulate Field, and Generalise Type?
A: No, I don't think so.
Q: Now, have you applied the tool support in the IDE you use for the refactorings you have used? Which IDE do you use?
A: We have IDE Eclipse as a standard. I think I have only ever done refactorings built into Eclipse.
Q: Why did you use only those?
A: I guess I just remembered them. And I'm pretty sure that they won't destroy my code. (laughs).
Q: What motivates you to use refactorings at all?
A: I like that they are so automated. It helps me save time. And they are a systematic way for me to improve my code.
Q: What would you need to use other refactorings that were not implemented in your IDE?
A: Good question. I have never thought about that. I guess I would need to be sure that I don't destroy some functionality. So, a good unit test suite would be necessary.
Q: Good. Thanks a lot for your time! I will send you my transcript and ask if I understood everything correctly. And of course, if you'd like, we will send you the final report of our study.
A: Yes, that would be nice. Goodbye!
Q: Bye!

Fig. 2 Sample made-up transcript for qualitative analysis exercise

paper-based coding exercises allow students to engage with the material directly and develop their skills without technological barriers. Using paper also fosters a more interactive and discussion-oriented classroom environment.

In conclusion, qualitative content analysis is a powerful tool for exploring complex social and environmental issues. Teachers can help students grasp this method and develop their analytical skills by providing clear examples and guidance. With

Table 1 This is an example output of the summarizing approach. Each line in the table shows the case it refers to, the page in the transcript, a short text paraphrasing one or a few sentences from the transcript, a generalization from the paraphrasing, and a reduction summarizing several lines in the table

Case	Page	Paraphrase	Generalization	Reduction
A	43	P1: Dev. was shocked by the many warnings presented to her after running the tool.	Overwhelmed by number of warnings	C1: Developer is overwhelmed by the number, diversity, and summaries from the static analysis report
A	44	P2: Dev. did not know where to start understanding the warnings from FindBugs	Overwhelmed by diversity of warnings	
A	44	P3: Dev. could not make sense of lengthy summaries	Overwhelmed by the summaries of warnings	

Table 2 This is an example of a coding guideline of deductive category assignment. It is prepared before coding. It shows the variable we want to analyze, the values the variable can take, the corresponding definitions of a value, and an example of when the value would be assigned to a text in the transcript. Finally, an additional rule is mentioned

Variable	Value	Definition	Anchor sample	Coding rules
Static analysis usage	U1: high level of usage	Frequent daily use of several static analysis tools	"I check every change with SpotBogs before I commit"	Both aspects must be met
	U2: medium level of usage	Use of some tool at least every week	"I apply Coverity before I present my work to the PO"	
	U3: low level of usage	Use of some tool from time to time	"We don't check our code regularly but before we release"	
	U4: usage not inferable	No clear statement from the developer	"We do have some static analysis tools"	

the right tools and approaches, researchers can uncover valuable insights into the data, leading to a deeper understanding of the issues at hand. Therefore, qualitative analysis is vital in case study research, mainly when dealing with small sample sizes or limited quantitative data.

You can find a more detailed treatment of teaching qualitative analysis in Chap. 9 by Treude. A detailed introduction of qualitative analysis could be included as part of case study research, or case study research could serve as an example of how qualitative analysis plays an important role when discussing analysis approaches.

6 Exercises for Students

There are at least three types of exercises that fit teaching case study research: (1) short exercises for lectures or tutorials, (2) case study critique, and (3) group projects. The short exercises are helpful to break up longer lecture sessions with interactive elements. They are suitable for smaller segments of 5–15 minutes, allowing students to reflect on and play with the knowledge and techniques they just heard about. Tasks most suitable for this are sketching a rough case study design for a concrete research question, writing an interview guide to collect data in a case study, and applying a qualitative data analysis technique just described to the students.

Here are three concrete examples of exercises:

- Let's assume that our experiment on the effects of specific refactorings did not show any statistically significant results. Yet many practitioners are advocating it. Hence, we want to understand its usage and impact in practice. Make a rough case study design.
- How could we apply these different data collection techniques in our refactoring case study? Develop an interview guide that you would use to interview developers in a company.
- We have a transcript of one interview with one developer regarding her use of refactorings in her last project. Try to do a summarizing qualitative analysis on it.

The case study critique takes longer and is more suitable for an intensive tutorial session. For this, select a recently published case study paper on a topic the students are likely familiar with and might even be interested in. Either give the students time in class to read the corresponding paper or assign this as homework to prepare for the tutorial. Then, dissect the paper in detail. Let the students map all details of the paper to the case study process and all necessary considerations. Emphasize all aspects that are well covered, and uncover all weaknesses in the research with the students. This generally repeats the case study process and increases the students' awareness of the finer details of designing and executing such a study.

Finally, a group project allows the students to go even deeper and apply this to their study. Letting the students pick their study ideas and research questions has been successful because this strongly increases their motivation to work on it. The teachers should, however, let the students present these ideas early and give feedback to ensure that the students are on the right track. Furthermore, hearing the other students present and giving feedback to them increases the learning effect for everybody. Then, the students work out the design of the case study in detail. Ideally, the case study is also executed, but this might not be possible in a university setting. An alternative is to let students generate artificial data to be further analyzed. The complete study with design, execution, and results is then again presented to the whole class. Again, discussing these group work results helps all students in the class better understand the details of case study research.

7 Pitfalls and Best Practices

Teaching case study research in software engineering can be challenging but rewarding. Here are some key pitfalls and best practices when educating students on this empirical method.

One common pitfall is the lack of clear objectives, as students often struggle without a well-defined research question or objective. Emphasizing the importance of starting with a clear and concise research question that guides the entire study is crucial. Another issue is an insufficient theoretical framework, which can result in case studies lacking depth and rigor. To address this, students should be taught to ground their case studies in relevant theories and models, ensuring a solid theoretical basis.

Selection bias is another significant challenge, where choosing cases that are not representative can lead to biased or non-generalizable findings. It is essential to strategically guide students in selecting cases to ensure they are representative and relevant to the research questions. Additionally, data overload can overwhelm students, leading to analysis paralysis. Therefore, students should be trained to focus on collecting data that directly addresses their research questions and to use data management techniques to stay organized.

Insufficient triangulation, where reliance on a single data source can result in biased or incomplete conclusions, is also a concern. Encouraging the use of multiple data sources, such as interviews, observations, and documents, helps triangulate findings and enhance validity. Weak analysis methods can undermine the validity of the findings, so it is crucial to teach robust qualitative and quantitative analysis methods, emphasizing coding, theme identification, and validation techniques.

Ethical issues must not be overlooked, as failing to address them can jeopardize the study's credibility and harm participants. Stressing the importance of obtaining informed consent, ensuring confidentiality, and adhering to ethical guidelines throughout the research process is imperative.

To mitigate these pitfalls, several best practices should be implemented. Providing a clear framework for designing and conducting case studies, including the development of research questions, case selection criteria, data collection methods, and analysis plans, is fundamental. Encouraging an iterative approach, where students continually refine their research questions, methods, and analysis based on initial findings and feedback, enhances the research process.

Using real-world examples and past case studies to illustrate key concepts and methodologies makes abstract principles more concrete and relatable. Highlighting the importance of integrating knowledge from other disciplines, such as sociology and psychology, enriches the understanding and interpretation of case study findings.

Promoting collaborative learning through group projects, peer reviews, and discussions enhances critical thinking and exposes students to diverse perspectives. Fostering a culture of critical reflection, where students regularly assess their assumptions, methodologies, and findings, ensures rigor and validity. Incorporating

technological tools for data collection, management, and analysis streamlines the research process and enhances analytical capabilities.

Providing continuous and constructive feedback throughout the research process helps students stay on track and improve the quality of their work. Encouraging students to publish their case study findings and present them at conferences fosters a sense of professional achievement and contributes to the broader research community.

8 Study Material

The online material[1] has a slide set covering this chapter's contents, including some of the exercises. In addition, the interview transcript from Fig. 2 is also available there as a separate text file to allow easier further usage.

9 Summary

In our experience, with this approach, students grasp the most important aspects of case study research, even in a classroom setting. The most important source of information on case study research in software engineering is the book by Runeson et al. [4]. Yet it is detailed enough to fill almost a whole course. Hence, you can only recommend it as a further reading or as a source for details for your teaching material.

References

1. Amershi, S., Begel, A., Bird, C., DeLine, R., Gall, H., Kamar, E., Nagappan, N., Nushi, B., Zimmermann, T.: Software engineering for machine learning: A case study. In: 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), pp. 291–300 (2019)
2. Mayring, P., et al.: Qualitative content analysis. Comp. Qualit. Res. **1**(2), 159–176 (2004)
3. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. Empir. Softw. Eng. **14**, 131–164 (2009)
4. Runeson, P., Host, M., Rainer, A., Regnell, B.: Case Study Research in Software Engineering: Guidelines and Examples. Wiley, Hoboken (2012)
5. Warrens, M. J.: Five ways to look at Cohen's kappa. J. Psychol. Psychother. **5**(4), 1000197 (2015)

[1] https://zenodo.org/doi/10.5281/zenodo.11544897

Teaching Design Science as a Method for Effective Research Development

Oscar Pastor, Mmatshuene Anna Segooa, and Jose Ignacio Panach

Abstract Applying design science research (DSR) methodology is becoming a popular working resource for most information systems (IS) and software engineering studies. The research and/or practical design problems that must be faced aim to answer the question of how to create or investigate an artifact in a given context. Precisely characterizing both artifact and context is essential for effective research development. While various design science guidelines and frameworks have been created by experts in IS engineering, emerging researchers and postgraduate students still find it challenging to apply this research methodology correctly. There is limited literature and materials that guide and support teaching novice researchers about the types of artifacts that can be developed to address a particular problem and decision-making in DSR. To address this gap in DSR, in this chapter, we explore DSR from an educational perspective, explaining both the concept of DSR and an effective method for teaching it. This chapter includes examples of DSR, a teaching methodology, learning objectives, and recommendations. Moreover, we have created a survey artifact intended to gather data on the experiences of design science users. The goal is to discover insights into contemporary issues and needs for DSR best practices (and, in consequence, evaluate the methodology we aim to teach). Our survey results offer a comprehensive overview of participants' experiences with DSR in the SE and IS engineering domain, highlighting broad engagement primarily initiated during PhD studies and driven by supervisory guidance and research problems. We finally disclose the artifact to the community

O. Pastor
Research Center on Software Production Methods (PROS), Universitat Politècnica de València, Valencia, Spain
e-mail: opastor@dsic.upv.es

M. A. Segooa (✉)
Informatics Department, Tshwane University of Technology, Faculty of ICT, Pretoria, South Africa
e-mail: segooama@tut.ac.za

J. I. Panach
Escola Tècnica Superior d'Enginyeria, Universitat de València, Valencia, Spain
e-mail: joigpana@uv.es

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2024

D. Mendez et al. (eds.), *Handbook on Teaching Empirical Software Engineering*,
https://doi.org/10.1007/978-3-031-71769-7_17

so that it can be used by other educators as a preparation when planning to teach DSR in tune with the experiences of their target audience.

1 Introduction

Design science research (DSR) has evolved rapidly as a common research methodology in the information systems (IS) and software engineering (SE) disciplines, which is aimed at designing innovative and practical information technology (IT) artifacts, such as conceptual models and software systems [1], through diverse approaches that make use of IS/SE engineering approaches such as innovative platforms, architectures, and technologies for the engineering of any specific kind of IS.

Natural science helps humans understand the world around them and seek explanations for the seemingly unexplainable. This is why IS studies frequently follow a natural science-oriented cycle, which suggests the structure that follows the steps of problem definition, literature review, hypothesis development, data collection, analysis, results, and discussion [4]. This is directly connected with the SE life cycle to be followed for the design and implementation of software systems for the IS under study. Our work has always connected IS development and SE practice through the proposal of conceptual modeling programming (CMP) approaches [3, 12, 13]. Following that working direction, in this chapter, we focus on a DSR application that can be applied to either IS or SE research working environments indistinctly.

According to Hevner [9], DSR is poised to take its rightful place as an equal companion to natural science research in the IS field. Therefore, IS scholars have realized the significance of DSR in improving the effectiveness and utility of IT artifacts when solving real-world business problems [8]. Wieringa [15] defines design science as the design or investigation of an artifact in a context that may follow either an engineering cycle to design an artifact or an empirical cycle to investigate a knowledge problem.

In this chapter, we adhere to Wieringa's perspective on DSR, which provides a clear and explicit distinction in characterizing research works. The distinction lies in either focusing on designing an artifact intended to solve a particular stakeholder problem, thereby achieving their goals (leading to a design cycle) or addressing knowledge questions that require empirical research studies (leading to an empirical cycle).

This initial step in DSR characterization is crucial to delineate whether the research deals with a practical problem, the resolution of which will "change the world" by creating something that did not exist before (the designed artifact). Alternatively, the research may aim to answer a knowledge question, contributing to extending knowledge about a specific problem without "changing the world" to create a new concrete artifact. This distinction is vital because the working cycle for

conducting a well-structured research project depends on this selection: practical problems versus knowledge questions.

Solving problems in a research-oriented manner needs to adhere to a design cycle strategy, while research aimed at addressing a knowledge question must follow an empirical cycle strategy. Since these two approaches involve different strategies, they also follow distinct steps. A design cycle encompasses problem investigation, treatment design, design validation, treatment implementation, and implementation evaluation. On the other hand, an empirical cycle involves an initial knowledge problem investigation, selection of the research design type, design validation, execution of the research (experiment), and evaluation of the results.

There is significant potential in advancing this domain by supporting emerging researchers with scientific materials that guide and teach them the application of DSR, considering its various dimensions, whether design-based or knowledge question-based, as a well-structured methodology. However, its application is not simple. Several significant decisions challenge the correct practice of DSR: Is the research problem centered around designing or improving an artifact (a practical problem), or is it, on the contrary, focused on answering knowledge questions? Is it a combination of both? In such cases, how does one integrate a design cycle, which involves creating an artifact, with its evaluation that requires result analysis to answer empirical research questions? Currently, our teaching experience in this context has identified gaps related to documenting the user experience to assess correct DSR practice. This serves as the central problem motivating our work.

All that leads to the problem of how to teach DSR correctly. What does it mean to apply DSR "correctly"? How do we identify the aspects that must be considered, and how do we evaluate the corresponding user experience? Moreover, how can these aspects be collected in a manner that can be used to gather practical data concerning DSR use? These problems have not received sufficient attention from DSR experts and are not well documented. Furthermore, there is a need for continuous assessment of DSR in IS and SE research. This chapter addresses also these problems through a study that aims to bridge the gap between DSR teaching and practice by presenting a survey for DSR assessment as a research methodology for IS and SE, which the DSR community may utilize for ongoing assessments.

The chapter's contribution is fourfold: First, it summarizes what DSR is, using an illustrative example. Second, it presents how to teach DRS, including learning objectives, thematic units, teaching methodology, teaching plan, and experience-based recommendations. Third, it proposes a concrete survey with the objective of gathering valuable data on the experience of DSR users to assess its practical use and discover good practice recommendations. Fourth, we report on selected results from running the survey from completed Master's and PhD theses at a university in Europe and a university in Africa.

This survey allows us to not only suggest further extensions of the DSR methodology but also clarify what aspects should be considered when teaching DSR in practice. The survey itself can be seen as a tool that, on the one hand, helps to better understand the particularities of DSR teaching and, on the other hand, can be

used also by others to generate a body of practical knowledge about how to improve design science-based teaching.

To achieve our objectives, we maintain consistency with our interest in DSR by following a Wieringa-based approach [15]. We begin by summarizing the DSR main characteristics and fundamentals in a Nutshell (Sect. 2), before discussing how to teach DSR (Sect. 3). Next, we state the problem and its need (problem investigation, Sect. 4) and introduce the survey artifact to analyze how DSR is perceived by students (Sect. 5). Subsequently, we report on the results and discuss them in Sect. 6, before concluding our chapter in Sect. 7.

2 Design Science Research: Fundamentals in a "Nutshell"

DSR can be defined as the design and investigation of artifacts in context [15]. The **artifacts** are designed to interact with a **problem context** in order to improve something in that context. The artifact itself does not solve any problem. It is the interaction between the artifact and a problem context that contributes to solving a problem. An artifact may interact differently with different problem contexts and solve different problems in different contexts. Examples of artifacts are methods, services, tools, business processes, or software systems, while examples of contexts are norms, desires, people, or organizations.

DSR is divided into two parts: **design problems** and **knowledge questions**. Design problems call for a change in the real world. A solution is a design, and there are usually many different solutions. Knowledge questions do not call for a change in the world but ask for knowledge about the world as it is. To see examples, let's use the title of a research work; that must be its best summary and must help to identify both artifact and context well.

For an example of a design problem, let's take this research work's title: "Design of a usability and usefulness test to evaluate the quality of cloud service providers." The artifact is the test itself, while the context is the quality evaluation of cloud service providers. An example of a knowledge question would refer to a research work intended to find out if the test is really usable and useful for cloud service providers, having as title "Evaluation of the usability and usefulness of a test to assess cloud service providers quality."

It is not always so simple to identify and distinguish artifacts and contexts. Look at this other example: imagine a research work with the title "From Adoption to Endurance: Exploring the Dynamics of AI Adoption Across Time and Contexts." While context is well identified (indeed, the whole title appears to introduce the context), what is the artifact in this case? How is the context under analysis going to create one solution? What solution? This is a good example of how teaching practices must take this crucial point as a central one for discussing good DSR practice and recommending research titles where both artifact and context are clearly identified, making true the statement that the research title must be the best summary of a research work.

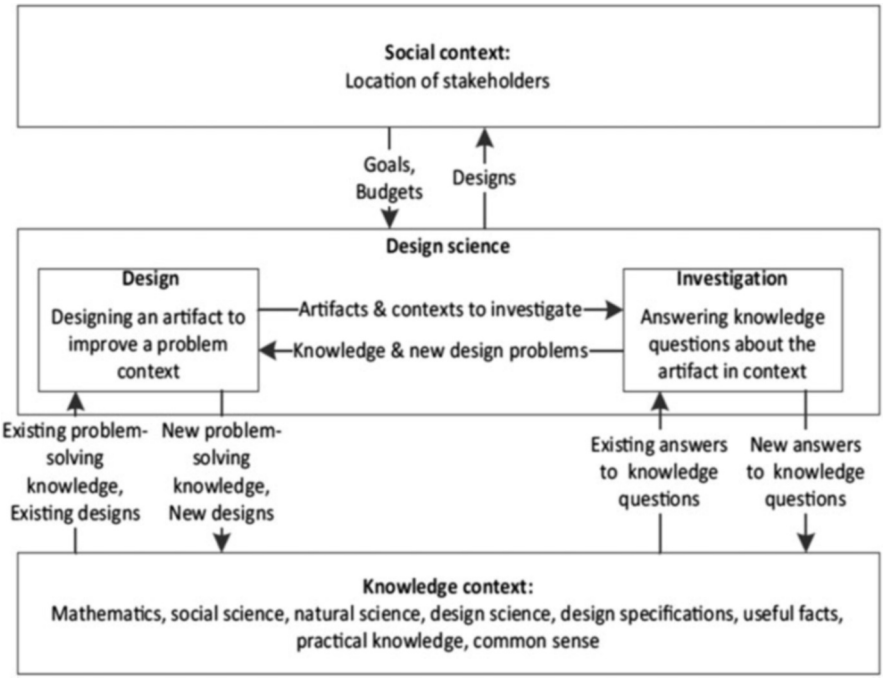


Fig. 1 A framework from design science [15]

Two different dimensions help to characterize the context correctly: social and knowledge. The problem context of an artifact can be extended to the stakeholders of the artifact and with the knowledge used to design the artifact. This extended context is the context of the DSR project as a whole. The resulting picture is a framework for design science, shown in Fig. 1, that should help to identify the context of a research work. The **social context** determines who are the stakeholders that will benefit from the use of the artifact, what their goals are, how they will be accomplished, or what are the funding constraints of a research project. The **knowledge context** consists of existing theories from science and engineering, specifications of currently known designs, valuable facts about currently available products, lessons learned from the experience of researchers in earlier design science projects, and plain common sense. DSR uses this knowledge and may add to it by producing new designs or answering knowledge questions.

For example, in the social context of genomic data analysis, we aim to design an artifact to help data analysts interpret the genomic information correctly. So, the investigation to answer is, "Is the system's usability enough to help improve the genomic data interpretation?" In the knowledge context, we find usability guidelines, templates of graphical user interfaces, and design recommendations.

DSR is conducted in a design cycle or in an empirical cycle. The **design cycle** is associated with a design problem. It includes the definition of the problem

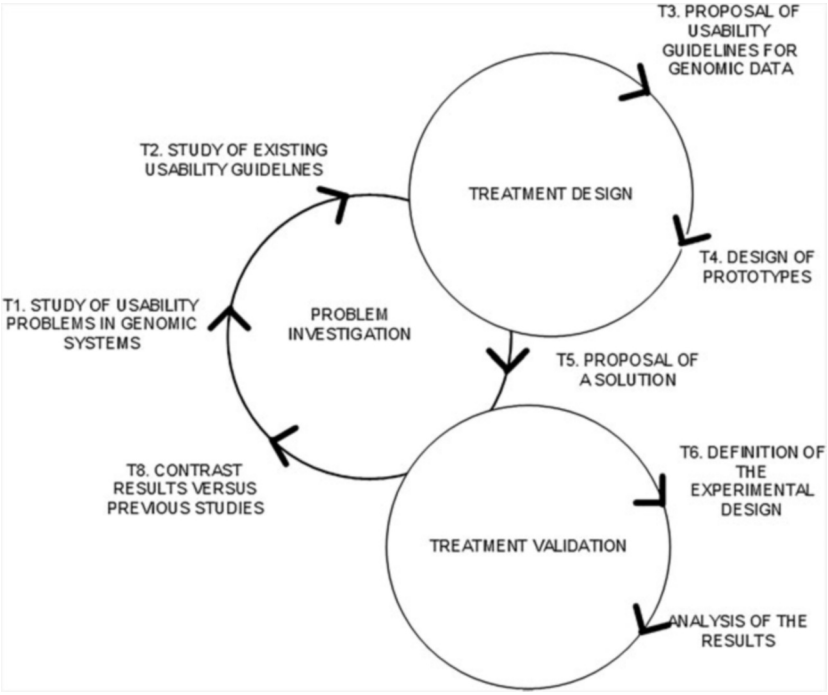


Fig. 2 Example of a design cycle to build usable systems in genomic data analysis

investigation, the treatment design, and the treatment validation. The result of the design cycle is a validated treatment ready to be transferred to the real world. Depending on the research strategy, the design cycle can be divided into different subcycles in order to have a large cycle that gathers all the research. Each subcycle must specify the different tasks to be fulfilled in order to reach the goals. For instance, this can happen if the research includes a sequence of component designs, and each one requires an individual design (sub)cycle. Or if each component of the design cycle has itself a cycle associated with its development.

Figure 2 shows an example of a design cycle divided into three subcycles for this specific illustration. This example is the one previously described to design an artifact to help data analysts to interpret genomic information. The tasks of each cycle are represented with the prefix "T" in Fig. 2. The first subcycle affects the problem investigation step. It includes tasks for studying the current state of the problem, the social context (or stakeholders), and the comparison of the results with existing works. The second subcycle is the treatment design, which aims to define artifacts in order to conduct the investigation. This cycle defines the knowledge context (or theories) of the research. The third subcycle is treatment validation, where empirical studies are conducted to analyze whether or not the results meet expectations.

The empirical cycle is associated with a knowledge question, and it is structured as a checklist of issues to decide when a researcher designs a research setup and wants to reason about the data produced by this setup. The cycle starts with a list of questions about framing the research problem. The rest of the checklist is about designing the research setup and the inferences from it, about research execution, and about data analysis.

The checklist of the empirical cycle is a logical grouping of questions that help you to find justifiable answers to scientific knowledge questions. The checklist depends on the specific research. An example of a checklist may include the motivation of the experiment, the research questions, the definition of variables, the specification of metrics, the definition of the experimental design, the definition of the protocol, and the mitigation of threats to validity.

This cycle aims to answer knowledge questions that cannot be answered using the literature, asking experts, or testing a prototype. The cycle includes tasks to define the research problem, design the research setup, research execution, and data analysis. In the same way, as in the design cycle, the empirical cycle can be divided into subcycles.

Figure 3 shows an example of an empirical cycle to analyze whether or not usability guidelines improve the usability of the graphical user interfaces and what

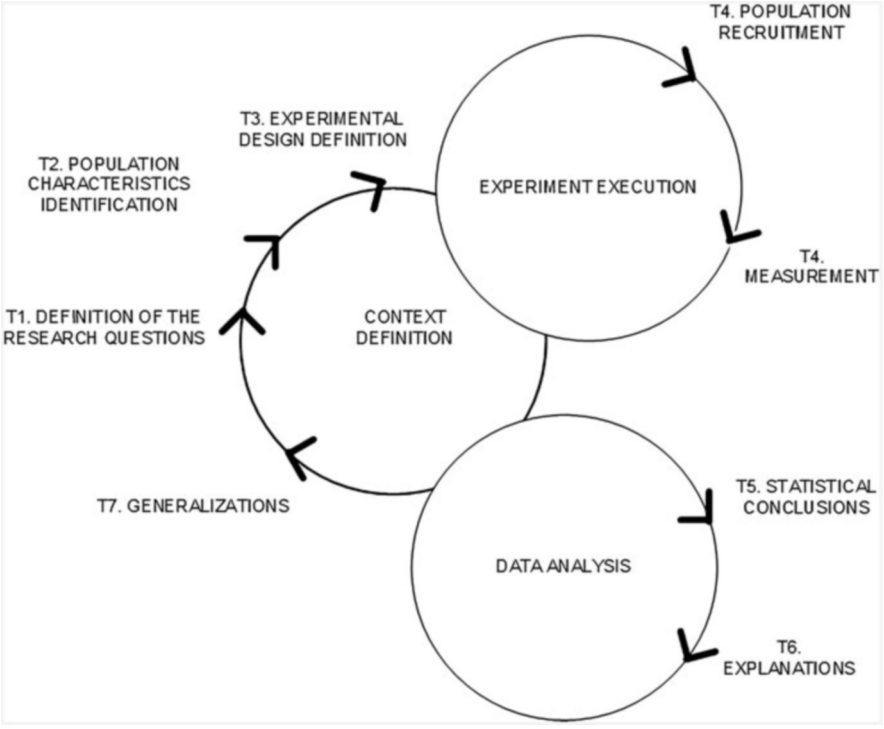


Fig. 3 Example of an empirical cycle to evaluate usability guidelines

guidelines have more effect on the user experience. Note that in this example, the research does not propose a new approach; it aims to evaluate existing approaches.

The example is composed of three subcycles, each one with specific tasks identified in Fig. 3 with the prefix "T." The first subcycle aims to define the context, which includes the definition of the research questions, the required profiles of the population, the experimental design, and the generalization of results once the experiment is over.

The second subcycle is the experiment execution, where we recruit the population and we conduct the experiment to get the measurement. The third subcycle analyzes the results with statistics and interprets such data.

Summarizing, four main aspects are to be considered relevant for a sound DSR teaching process which educators may take into consideration when formulating DSR learning outcomes:

1. The first aspect to take into account is how to identify the precise artifact that is designed and the proper context in which it is intended to work. The correct identification of both artifact and context becomes an essential initial step of any DSR process, as we have already discussed earlier.
2. Another aspect to consider is selecting an adequate title, where both the artifact and the context can be clearly recognized.
3. A third aspect is to delimit the scientific nature of the research. Does the research core face a design problem or a knowledge question? Is there a new artifact as a result of the research work that did not exist before? Or, on the contrary, the research work is about accumulating knowledge concerning already existing artifacts?
4. Once the research nature has been characterized, the fourth and last aspect refers to selecting the particular research cycle (design cycle or empirical cycle) that better fits the planned research work that must be developed.

These four aspects and considerations of how DSR is used in practice, together with all the dimensions discussed in the previous sections, have guided the learning objectives and thematic units that should structure a basic course to teach adequately Design Science. The next section describes how to teach DSR, covering both the theoretical foundations and the practical skills necessary to engage in this innovative research methodology.

3 How to Teach Design Science Research

This section describes how to teach DSR in a potential course for researchers in IS and SE, especially for PhD students who are starting their research. The recommendations shared in this section are rooted in our experiences of teaching DSR in our own educational environments and tutorials given in the research community, and they should also be understood as such experience-based recommendations. We elaborate on the course's objectives, how it is divided into thematic units, the

teaching method, and the teaching plan, followed by practical recommendations and advice. Classes should combine theory and practical exercises, ensuring that all theoretical concepts can be seen in real-world applications.

We begin by outlining the learning objectives that students are expected to achieve upon completing a course:

- **Objective 1:** Students should be able to define and explain key concepts of design science.
- **Objective 2**: Students should demonstrate the ability to apply design science principles to real-world problems.
- **Objective 3:** Students should effectively work in teams, contributing to and learning from their colleagues.
- **Objective 4:** Students should exhibit strong critical thinking skills, be able to evaluate their work, and make improvements based on feedback.
- **Objective 5:** Students should be able to integrate design science principles into their PhD or MSc research, producing high-quality, rigorous academic work.

In order to reach such objectives, the course is divided into **thematic units**. Thematic units have been extracted from the book written by Wieringa [15], which describes how to apply DS to computer science in detail.

- **Unit 1:** What is design science?
- **Unit 2:** Research goals and research questions
- **Unit 3:** The design cycle
- **Unit 4:** The empirical cycle

These thematic units are taught using a teaching method that begins with the problem definition. This involves identifying a problem or need within the realm of SE, such as developing new software, optimizing an existing system, or creating a new software development method, among others. This problem will be used throughout the entire teaching process. The teacher defines the problem on their own, and it is recommended to choose a problem with which most of the students are familiar. Moreover, the teacher divides the students into several groups of three or four students. These groups will participate in the following steps of the teaching method.

Once the problem has been defined, the method involves describing all the elements in a theoretical class. All the elements shown in Fig. 1 are explained. Each concept is defined as described in the definition of the DSR method. Next, using the previously defined problem, each group of students must think about how the concept can be operationalized for that specific problem. For instance, this is the right time to determine if the research is centered around a practical problem, a knowledge question, or both. A precise answer to these questions will facilitate the correct DSR application process. First, there is an initial discussion within each group. After 5 minutes of internal discussion, each group shares their thoughts with the other groups. Each concept involves brief brainstorming within the groups and among the groups, facilitating the generation of ideas. At the end, the teacher decides which ideas are the most suitable for that problem.

After introducing all the elements of the DSR method, and once the nature of the research is delimited (practical problem, knowledge question, and their potential connections), the next step is to consider the necessary cycles that depend on the research nature decision. Each group must determine whether the research method requires one or more cycles, which of these cycles are to define the approach (design cycle), and which ones are required to validate those approaches (empirical cycle). Additionally, each team must specify the different tasks required in each cycle. After one hour of group work, each group shares their conclusions with the other groups. The teacher decides which proposals are the most suitable for that specific problem, discussing the pros and cons of each solution. Once the concepts of the DSR method have been introduced and students know how to define the different cycles, the final part of the teaching method consists of applying the DS method to each student's PhD thesis. Groups are eliminated, and each student works individually. Usually, the DSR course is taken by PhD students, so each student has a different research project where DSR can be applied. From now on, we assume that the audience of the proposed course are PhD students, although the ideas can be applied to any academic research working context.

Working individually, each student must specify how the different terms of DSR can be applied to their research, as well as think about the necessary cycles and tasks. A short document must be prepared to show specifically how DSR can be applied. Elaborated documents are reviewed by the teacher outside the classroom. Written feedback can be provided to each student in such a way that, beyond learning DSR, the result can be included in the PhD document as part of the student's work.

The teaching plan should span at least 4 hours to be able to teach the minimum while it can be gradually expanded. In that sense, we propose at least two lecture units or a half-day tutorial to teach the essence of design science. Effectively teaching DSR can as well be captured in an entire course that could emphasize particular aspects of the practical application of Design Science depending on the setting and which learning objectives the educator might want to stress. Next, we describe the **teaching plan** considering the minimum time we recommend to spend on key aspects (4 hours in total). In that sense, a course can also be effectively applied as part of tutorials and adopted to the corresponding formats. Below is the timeline:

- **From 0 minutes to 15 minutes**: The teacher presents the problem to be discussed during the class and creates the groups of students.
- **From 15 minutes to 1 hour and 30 minutes**: The teacher presents the different elements of DS. Each element is worked on within each group and then shared among the other groups.
- **From 1 hour and 30 minutes to 2 hours and 30 minutes**: Each group must specify the different cycles to solve the described problem. There is a discussion first within each group and then among the groups.
- **From 2 hours and 30 minutes to 4 hours**: Each student must individually apply DSR to their PhD research and elaborate a short document with the results.

Finally, we summarize some **recommendations** both from the point of view of the teacher and from the point of view of the student. From the point of view of the teacher:

- **Choose real problems**: Select problems that are relevant and familiar to the students. This enhances engagement and makes the learning process more meaningful.
- **Group work**: Encourage collaborative learning by organizing students into groups. This fosters teamwork and allows for the exchange of diverse ideas. The results of the group work do not need to be perfect but should serve as a way to discuss several alternatives.
- **Regular feedback**: Provide timely and constructive feedback throughout the learning process to guide students and help them improve their work. It is recommended to provide written feedback individually to each PhD thesis. This way, all details can be processed calmly by the student.
- **Problem used as a guide throughout the learning process**: Choose a problem that is easy enough for all students to understand the context and the problem itself. It is not advisable to start with a problem related to a real PhD thesis, as these topics are very specific and not all students may understand them. Try to adjust the problem to the audience.
- **Encourage critical thinking**: To properly apply DSR to each PhD research, the student must have learned DSR properly beforehand. The teaching process should strongly involve discussion both within the groups and among the groups. If the students do not participate actively in the discussions, the teacher must encourage and facilitate this discussion. It is crucial that students learn from the discussion what is correct and what is not. In classes with poor discussion, more errors are observed when students apply DSR to their PhD research.

From the point of view of the students, these are the recommendations:

- **Participate actively in the group work**: Collaborate with your peers in group activities. This fosters teamwork, allows for the exchange of diverse ideas, and helps you learn from different perspectives. Remember, the goal is not perfection but to explore various alternatives and learn from the process.
- **Seek and reflect on feedback**: Regularly seek feedback from the teacher and colleagues of the group. Constructive feedback is crucial for your improvement. This can, for instance, help to clearly identify what is provided by the research (concrete artifact) and what is the context of interest for such a solution, which are crucial dimensions for conceptually characterizing a research contribution. When the feedback of the PhD research is provided at the end of the course, take time to reflect on it. DSR properly applied to your research is an essential section in your PhD memory.
- **Engage in discussions**: Actively participate in discussions during class. Whether in group discussions or class-wide debates, sharing your thoughts and listening to others will deepen your understanding. Do not be afraid to speak up even if you are unsure, and these discussions are valuable learning opportunities.

- **Document your process**: Keep detailed records of your design process. Documenting your steps, decisions, and reflections will help you track your progress and provide valuable insights for future projects. Taking notes and ideas in the discussion stage can be very helpful when you apply DS to your specific research. Words easily go with the wind.

4 Problem Investigation

Despite the fact that more than a decade of DSR discussions and the publication of the seminal articles by Gregor et al. [6], Hevner et al. [10], Kuechler and Vaishnavi [11], Peffer et al. [14], and Wieringa [15], which led to significant growth in DSR popularity in IS and SE, good DSR practice is still a problem for researchers interested in using DS as their research methodology. Literature highlights the emerging development of DSR methodology in postgraduate studies projects, which presents a challenge to understanding their research philosophy and the application of this methodology [2].

The reality is, however, that the experience of emerging researchers, such as doctoral students, regarding DSR adoption and application is still not well documented [2, 5]. This renders the design of appropriate education and training cumbersome. Furthermore, there is limited support and guidance for documenting and effectively managing DSR processes. This indicates that the existing literature and materials that guide novice researchers on the underlying philosophy of DSR studies, the development of artifacts to address specific problems, and decision-making on DSR guidelines to address issues need to have robust and well-documented support.

Given this shared background, it is evident that evaluating the application of DSR to improve the user experience and support the methodology constitutes a challenging journey. This justifies the development of a standard survey artifact that may serve as a tool for DS custodians such as [6, 7, 10, 11, 14, 15] and IS and SE engineering researchers in their ongoing work. The survey should include a set of questions that conceptually characterize the dimensions to be considered if we want to understand DSR practice. We want to improve how to teach it by identifying which aspects the teaching process should focus on, and we aim to collect data for further analysis.

The data collected using our survey artifact may contribute theoretically to the DSR literature gap (conceptual characterization of DSR methodology versus its successful practical application) by providing insights into the DSR user experience and the need for improvements in this domain. Lastly, the analysis results can also aid in assessing the maturity of the DSR methodology in solving real-life problems to deliver valuable artifacts in concrete contexts, providing value in a clear, well-delimited, and convincing way. All these objectives guide our work, leading to the design of the survey that we introduce in this work.

5 Survey Artifact for DSR Users' Experience

There is no guide to applying DSR correctly, and the practical application of some elements of the DSR teaching process (which we discuss in this chapter) can be confusing. To develop a solution to this practical DSR problem, we propose a version of a survey artifact that intends to gather data on design science users' experience to assess the methodology application and discover insights on the contemporary issues and needs for DSR best practice. It should go without saying that this artifact may be further revised in the future to meet the individual needs of other scholars. In that sense, our survey may be re-used and adopted by other educators as a preparation for designing/aligning the course with the target audience.

The survey draft was informed by the literature on the DSR research gap and the need to scientifically document DSR user experience, especially for the case of postgraduate students, with the aim to discover insights that will help DSR custodians and scholars in the IS engineering and SE domains to provide the necessary support to apply best practices toward the methodology. Our survey, in its basic version, has 20 questions as indicated in Table 1. These questions are clustered into six groups, whose conceptual justification is now introduced.

- Cluster 1: Experience: questions 1 and 2 focus on identifying the level of DSR experience as a scholar in the domain.
- Cluster 2: Purpose: questions 3–5 delimit the problem concerning the artifact intended to design (artifact identification-oriented).
- Cluster 3: DSR guidelines selection: questions 6–9 determine the DSR guidelines selected to accomplish the artifact's design in its working context (context characterization-oriented).
- Cluster 4: Evaluation: questions 10 and 11 explore if the artifact has been evaluated and how.
- Cluster 5: User's experience: questions 12–19 investigate the support, experience, and satisfaction in the application of the DSR guidelines.
- Cluster 6: Dissemination: question 20 determines if the results of the project delivered through DSR have been published, emphasizing the sharing of information about DSR use and value.

6 A First Qualitative Analysis of the Preliminary Survey for DSR Users' Experience

In our attempt to employ the proposed survey to collect data that could help us to better understand and improve teaching design science, this section presents preliminary results obtained from the qualitative analysis of the survey. It is reported according to the six clusters that structure it. The analysis was conducted on a small

Table 1 Survey artifact for DSR application and experience, including a first set of "yes/no, multiple choice" and "open-ended" questions

1.	Have you used design science research prior PhD?
2.	Where did you use design science methodology?
3.	What motivated the research to take the DSR approach?
4.	What problem was the research solving? Or what was the research problem?
5.	What type of artifact has the study produced?
6.	Whose design science guidelines did your study follow?
7.	Were you aware of other guidelines than the one you used for your study?
8.	If yes, why did they go for the one you chose, if you selected no then type N/A.
9.	Did the main purpose of your study follow the empirical research cycle and/or engineering cycle?
10.	Did you evaluate the artifact?
11.	How did you evaluate the artifact?
12.	Did you attend a design science workshop before/during the study?
13.	What challenges have you experienced when using design science methodology?
14.	What opportunities have you experienced when using DSR methodology?
15.	What are the lessons learned in the journey?
16.	What kind of support do you wish you had received during the journey?
17.	Were they ever discouraged from carrying on with the methodology during the study?
18.	What is interesting about the methodology?
19.	What is less interesting about the methodology?
20.	Have you published your research conducted using design science methodology?

sample size, as it was intended to pilot the artifact in preparation for data collection from a larger sample.

Our sample included participants who utilized DSR in their postgraduate studies and who were in the scope of educational measures taken in order to prepare them for applying DSR.

Responses were received from 21 participants (4 doctoral, 14 master's graduates, and 3 bachelor's degrees). Feedback is useful for analyzing how DSR is applied in various research fields. Moreover, the results provide insights into the benefits and drawbacks of using DSR in a teaching context and indicate how to proceed to continuously improve such a DSR teaching experience.

6.1 Artifact Results and Discussion

This section introduces and discusses cluster by cluster on what the survey results meant in terms of the better understanding and continuous improvement of the

teaching methodology that is to be followed for DSR teaching practice. In responding to the questions in cluster 1, which aimed to gather the DSR experience of the participants as scholars in the IS engineering domain, two questions were posed. These questions were designed to identify users' experience and knowledge of DSR. Q1 inquired about prior DSR use (before either a PhD thesis or an MSc thesis), while Q2 focused on where participants had utilized design science methodology before.

The participants indicated that they have used DSR of which most of them highlighted that they used the methodology before their PhD studies. We conclude that it is important to know the DSR background of DSR users before starting a sound DSR teaching strategy to calibrate well how detailed the introductory material must be.

Cluster 2, which aimed to define the problem that the artifacts were to address, was faced using three questions identified as Q3 to Q5 that focus on the survey instrument. Q3 inquired about the motivation behind choosing the DSR approach; Q4 sought to determine the problem the research aimed to solve or the research problem itself; and Q5 was designed to explain the type of artifact the study intended to generate.

The findings highlight that the motivation to conduct the study through design science came from the supervisor/study advisors, while some participants followed the methodology to better characterize the problem that the study intended to address. The artifact that most participants produced was a model, system, or method.

In this context, an accurate artifact identification becomes an essential actor of the DSR teaching process. This confirms our practical experience, where the precise identification of the artifact that a research work is designing or informing is what takes more time and stronger discussions. A research work must be built on a clear and concrete characterization of the artifact that is being designed.

Problems which were solved by the DSR studies included.

- Genomic data quality measurement.
- Design, develop, and validation of a platform for managing genomic data.
- Conceptual model of the genome to improve understanding of the genomic domain.
- Creating artifacts to teach systematic literature review.
- What model will inform the adoption of a smart learning environment for a secondary school in Ekurhuleni North district?
- Lack of appropriate models to inform data governance relating to Big Data.
- Access to education in low bandwidth areas.
- Online fraud detection model.
- How can a pharma company integrate Gen AI to stay competitive and innovative?
- Developing a method for identifying KPIs for a strategic alignment of IT projects.
- How to diminish the impact of senior consultant turnover on knowledge management in a software editing company?
- What strategies are most effective in managing organizational change when transitioning to electronic invoicing in B2B processes?

Table 2 Examples of DSR problems and cycles

Examples of DSR problems	DSR cycle	Type of DSR problem
Genomic data quality measurement	Engineering	Design
Design, develop, and validation of a platform for managing genomic data.	Engineering	Design
Conceptual model of the genome to improve understanding of the genomic domain	Engineering	Design
Creating artifacts to teach systematic literature review	Empirical	Knowledge
What model will inform the adoption of a smart learning environment for a secondary school in Ekurhuleni North district?	Empirical	Knowledge
Lack of appropriate models to inform data governance relating to Big Data	Empirical	Knowledge
Access to education in low bandwidth areas	Empirical	Knowledge
Online fraud detection model	Empirical	Knowledge
How can a pharma company integrate Gen AI to stay competitive and innovative?	Empirical	Knowledge
Developing a method for identifying KPIs for a strategic alignment of IT projects	Engineering	Design
How to diminish the impact of senior consultant turnover on knowledge management in a software editing company?	Empirical	Knowledge
What strategies are most effective in managing organizational change when transitioning to electronic invoicing in B2B processes?	Empirical	Knowledge
Rethinking/redesigning the requirements engineering process of an IS department	Empirical	Knowledge
What solutions can be implemented in a company to reduce the costs of IS?	Empirical	Knowledge
How can organizations implement strategies to increase user adoption of SSBI tools?	Engineering	Design

- Rethinking/redesigning the requirements engineering process of an IS department.
- What solutions can be implemented in a company to reduce the costs of IS?
- How can organizations implement strategies to increase user adoption of SSBI tools?

For better understanding when we face a design (practical) problem or a knowledge question, with their corresponding engineering cycle (in the design problem case) or the empirical cycle (for the knowledge question case), Table 2 shows the list of DSR problems that we have presented above, classified according to these characteristics.

From the perspective of the survey's assessment, it was positive to observe that very different topics aligned well with the survey's purpose, and that identifying the type of artifact at the core of the analyzed research became a clear and informative

task. The problem identified by the survey reflects interest in both the design cycle and the empirical cycle. Some topics focused on design (practical) problems, where models and methods were developed. In contrast, others focused on knowledge problems, involving information-based investigation into how organizations can better implement strategies and studies on how to diminish the impact of senior consultant turnover. Our last conclusion in this context was how important it is to clearly define if the DSR process involves a practical problem or a knowledge question. Again, this confirms our practical experience around how to determine this aspect after having precisely identified artifact and context: artifact and context require a different treatment depending on being facing a practical problem or a knowledge question, as it has been discussed before in this chapter.

Cluster 3, where DSR guidelines were selected, aimed to determine the guidelines used to accomplish the artifact's design. In this cluster, four questions were posed on questions 6–9 of the survey instrument. Q6 asked about which DSR guidelines were followed in the study; Q7 inquired whether participants were aware of different DSR guidelines than the ones they used for their study; Q8 complemented the previous question, clarifying why the participant selected the chosen guidelines when the answer to Q7 was yes. Finally, Q9 asked about the main purpose of the study when following either an empirical research cycle and/or an engineering research cycle.

The results indicate that participants commonly used DSR guidelines from Hevner et al. [10], Vaishnavi and Kuechler [11], Peffers et al. [14], and Wieringa [15]. Those who used Wieringa [15] tended to follow the engineering cycle, while participants who used Hevner et al. [10] and Peffers et al. [14] followed the empirical cycle. Moreover, those who followed Wieringa [15] were not aware of other guidelines, while those who used Hevner et al. [10], Peffers et al. [14], and Vaishnavi and Kuechler [11] were knowledgeable about alternative DSR guidelines.

Participants noted the reason for choosing Hevner et al. [10] as its ease of understanding and support for model development. On the other hand, the choice of Peffers et al. [14] was attributed to the clarity and detail of its guidelines, along with the recommendation by supervisors.

Based on the questions asked in cluster 3 and the responses from the participants, we found that answering the questions became an easy task, and the provided information was useful for better understanding and comparative analysis.

We assume that DSR can be successfully applied using any of the existing DS approaches, and our conclusion here is that it is important to make DSR users be aware of the fact that different valid DSR approaches exist and that it is important to make explicit and known their strong conceptual connections. It is also fruitful to explain why a particular selection is done to be used in practice.

Cluster 4, which focused on the evaluation of the artifact, aimed to establish if the artifact was implemented and how the evaluation was conducted. To address this cluster, two questions were presented through Q10 and Q11 of the survey instrument. Q10 inquired about whether the research work included artifact evaluation, while Q11 focused on how the artifact had been evaluated. From the findings, all the participants indicated that their artifacts were evaluated. The evaluation was

done through experiments, surveys, focus groups, and interviews. It appears that all who used the engineering cycle of the DSR used experiments and interviews to evaluate the artifacts of their study, while those participants who followed the empirical cycle used surveys and focus groups.

Based on the questions asked in cluster 4 and the responses from the participants, we also found the results informative, aiding in obtaining a better understanding of what was done and how, particularly in terms of distinguishing the practical problem vs knowledge question dichotomy that we discussed earlier. The main implication of this distinction is that a design cycle focuses on the design of an artifact, and an implementation directly suggests the use of such a practical problem-based working context. In this case, the evaluation corresponds to a validation step, and it is then a part of the contribution.

Alternatively, an empirical cycle indicates that there is no implementation, and the value of the research concentrates on a strong experimental task that constitutes the core of the research contribution. An in-depth analysis of particular survey answers can allow to assess that these connections exist in order to confirm that a sound DSR practice is applied. On the other hand, accumulating data from many individual cases could make it possible to perform a data analytics process to confirm that correct DSR patterns appear when we compare a significantly diverse set of concrete design science applications cases.

Cluster 5 aimed to gather users' DSR application experience to investigate the support, experience, and satisfaction in the application of DSR guidelines. It covers the user's experience dimension. This cluster was addressed through eight questions on the survey artifact, specifically questions 12–19. These questions cover various aspects in this regard. Q12 explores whether participants attended any design science workshops before or during the study; Q13 asks about challenges experienced by DSR users when using the design science methodology; Q14 seeks declarations of opportunities obtained as a result of DSR methodology use; Q15's purpose is to find out lessons learned in the DSR application journey; Q16 detects any potential support that was lacking during the process; Q17 asks for potential discouragement from continuing with the methodology during its application; Q18 emphasizes aspects considered as more interesting by participants and that provide a concrete, useful benefit; finally, Q19, on the contrary, explores what was seen as less interesting about the methodology.

This cluster provided interesting information that showed us how collecting this type of info could help us better understand the particular context of DSR practice and what affects the planning of a DSR teaching process. In particular, the participants reported to have received training (except for one participant who indicated that they did not receive training during their studies). Understanding the design cycles seemed to be a challenge for the participants who did not receive training, emphasizing that the methodology is highly encouraged. Furthermore, other participants indicated that they experienced challenges in positioning the research problem into either design or empirical problems and selecting the evaluation framework. This mimics our practical experience, where the characterization of

the research work as either a research (practical) problem or knowledge (empirical) question is always a complex task for students.

Therefore, a suggestion to provide more examples of the methodology is recommended to support students in understanding DSR. Additionally, training should be offered at the early stages of the study. Educators can also design the learning material based on the participant's needs, and clear learning objectives and outcomes may be set to address these challenges.

Some participants highlighted that they were initially afraid to use DSR; however, they realized it was doable as they persevered and managed to complete their studies. Overall, the participants indicated they were not discouraged from using DSR, although some mentioned feeling discouraged along the journey as they engaged with the methodology.

This cluster facilitated the identification of methodological problems encountered by the participants in practice. The information was more diverse, yet informative, covering an aspect—methodological usability and the practical ease of use—whose positive properties were not as clear and convincing as expected.

Cluster 6 aimed to establish if the work had been disseminated. Specifically, question 20 determined if the project's results delivered through DSR had been published, emphasizing the sharing of information about DSR use and value. To address this cluster, the question was whether participants have published the research conducted using design science methodology to determine if the DSR use was reported and documented. Out of 21 participants, only 19% (four participants) have published their work; some published in conference proceedings, while other participants published their work in journals.

Summarizing the survey experience, the results provide a comprehensive overview of the participants' experiences and insights regarding the application of DSR in the SE and IS engineering domains. The responses indicate a broad engagement with DSR methodologies, primarily initiated during PhD studies, driven by both supervisory guidance and the inherent problems the research aimed to solve. Evaluation methods varied, with empirical cycles favoring surveys and focus groups, while engineering cycles leaned toward experiments and interviews. Training and methodological clarity emerged as critical needs, with participants emphasizing the value of early and structured educational support. Despite initial apprehensions, participants found the DSR methodology feasible and ultimately rewarding, with most publishing their findings. Overall, the survey underscores the practical utility and adaptability of DSR while highlighting areas for enhanced support and training to optimize its application in academic research.

7 Conclusion

This work has introduced DSR to identify what aspects must be considered when improving and reinforcing DSR teaching in practical cases. We introduced a set of illustrative examples and discussed them from an educational point of

view. Our work also describes the learning objectives of a DSR course, the teaching methodology, thematic units, and the teaching plan before summarizing recommendations. Next, we introduced a survey artifact that can be used to gather reflections and experiences of DSR users, especially emerging researchers. The survey incorporates those aspects we consider crucial for a good DSR teaching practice, and it is intended to assist in the ongoing assessment of DSR in SE and IS engineering to attract expert guidance and interventions required for achieving a sound DSR teaching process. We encourage the community to make use of this survey artifact and contribute, with its adoption, to generate further valuable data for the community to better understand how DSR is used in practice.

Our own survey instance has already provided a useful set of lessons learned that are presented in this chapter. We encourage further adoption and application by the community to gather valuable data that should help fine-tune the DSR teaching process we have proposed. Our own immediate working plan includes conducting a massive data-gathering process intended to collect information on DSR practical use in several universities, oriented toward PhD and MSc students that use design science as a research methodology for their research. The survey aims to be naturally improved with the results of such evaluations. Educators may use the survey to assess learners' experience with the DSR application; moreso, gaps may be identified, which will further inform the DSR learning, design, and materials required for the associated teaching process.

Online Material

The survey artifact used in this chapter can be found in the online material section under https://zenodo.org/doi/10.5281/zenodo.11544897.

Acknowledgments We would like to acknowledge the participants in the pilot study to assist in evaluating the draft artifact. Their support contributed to the improved artifact, which will be used to collect the main study data from a large sample of participants. This work was developed with the support of the Spanish Ministry of Science and Innovation, co-financed by FEDER, in the project SREC (PID2021-123824OB-I00).

References

1. Brocke, J.V., Riedl, R., Léger, P.M.: Application strategies for neuroscience in information systems design science research. J. Comput. Inf. Syst. **53**(3), 1–13 (2013)
2. Cater-Steel, A., Toleman, M., Rajaeian, M.M.: Design science research in doctoral projects: an analysis of Australian theses. J. Assoc. Inf. Syst. **20**(12), 3 (2019)
3. Embley, D.W., Liddle, S.W., Pastor, O.: Conceptual-model programming: a manifesto. Handbook of conceptual modeling: theory, practice, and research challenges, pp. 3–16. Springer, Heidelberg (2011)
4. Geerts, G.L.: A design science research methodology and its application to accounting information systems research. Int. J. Acc. Inf. Syst. **12**(2), 142–151 (2011)

5. Gerber, A., Kotze, P., Van der Merwe, A.: Design science research as research approach in doctoral studies. In: AMCIS 2015 Proceedings, Puerto Rico, 13–15 Aug 2015
6. Gregor, S., Jones, D.: The anatomy of a design theory. J. Assoc. Inf. Syst. **8**(5), 312 (2007)
7. Gregor, S., Hevner, A.R.: Positioning and presenting design science research for maximum impact. MIS Q., 337–355 (2013)
8. Hevner, A., Chatterjee, S.: Design research in information systems theory and practice. Springer, Heidelberg (2010)
9. Hevner, A.R.: A three cycle view of design science research. Scand. J. Inf. Syst. **19**(2), 87–92 (2007)
10. Hevner, A.R., March, S.T., Park, J., Ram, S.: Design science in information systems research. MIS Q. **28**(1), 75–105 (2004)
11. Kuechler, B., Vaishnavi, V.: On theory development in design science research: anatomy of a research project. Eur. J. Inf. Syst. **17**(5), 489–504 (2008)
12. Pastor, O., Molina, J.C.: Model-driven architecture in practice: a software production environment based on conceptual modeling, vol. 1. Springer, New York (2007)
13. Pastor, O., Gómez, J., Insfrán, E., Pelechano, V.: The OO-Method approach for information systems modeling: from object-oriented conceptual modeling to automated programming. Inf. Syst. **26**(7), 507–534 (2001)
14. Peffers, K., Tuunanen, T., Rothenberger, M.A., Chatterjee, S.: A design science research methodology for information systems research. J. Manage. Inf. Syst. **24**(3), 45–77 (2007)
15. Wieringa, R.J.: Design science methodology for information systems and software engineering. Springer, Heidelberg (2014)

Teaching Survey Research in Software Engineering

Marcos Kalinowski, Allysson Allex Araújo, and Daniel Mendez

Abstract In this chapter, we provide advice on how to effectively teach survey research based on lessons learned from several international teaching experiences on the topic and from conducting large-scale surveys published at various scientific conferences and journals. First, we provide teachers with a potential syllabus for teaching survey research, including learning objectives, lectures, and examples of practical assignments. Thereafter, we provide actionable advice on how to teach the topics related to each learning objective, including survey design, sampling, data collection, statistical and qualitative analysis, threats to validity and reliability, and ethical considerations. The chapter is complemented by online teaching resources, including slides covering an entire course.

1 Introduction

Survey research has become an indispensable tool in software engineering, allowing for gathering cross-sectional insights into opinions, experiences, and needs with respect to practices and problems that can help shape the discipline. In this chapter, we take a particular focus on addressing the challenges we experienced when teaching survey research in software engineering, aiming to provide a comprehensive guide that educators can follow and adapt to their specific contexts.

Drawing from several international teaching experiences among the authors, we start by providing teachers with a comprehensive overview of a what we ourselves

M. Kalinowski (✉)
Pontifical Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil
e-mail: kalinowski@inf.puc-rio.br

A. A. Araújo
Federal University of Cariri, Juazeiro do Norte, Brazil
e-mail: allysson.araujo@ufca.edu.br

D. Mendez
Blekinge Institute of Technology and fortiss, Karlskrona, Sweden
e-mail: daniel.mendez@bth.se

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2024

D. Mendez et al. (eds.), *Handbook on Teaching Empirical Software Engineering*,
https://doi.org/10.1007/978-3-031-71769-7_18

consider to be a typical syllabus for teaching survey research. This syllabus includes a detailed course description, clearly defined learning objectives, a schedule of lectures to be taught, and examples of practical assignments. Please note that we do not claim (nor intent) to provide structure for a universal way of teaching survey research. Rather, we describe a blueprint that we extracted from our own past experiences. Those experiences emerge from both teaching survey research in various settings (MSc level and PhD level) and applying the core concepts as part of large-scale surveys and families of surveys (e.g., [2, 15, 19, 22, 23, 32, 45]). We trust that the reader will know how to adapt the insights given to the particularities of their context and hope to support them in this endeavor with the selection of topics that we believe are important to cover.

The chapter provides advice for proficiently teaching the main topics related to survey research. These topics include the characteristics and purpose of survey research, survey design, sampling, data collection, statistical and qualitative analysis, threats to validity and reliability, and ethical considerations. Furthermore, the chapter is complemented by online teaching resources, which include slides covering all the lectures. Those resources can be found in the online repository that accompanies this edited book: https://zenodo.org/doi/10.5281/zenodo.11544897.

It is noteworthy that while the chapter offers a comprehensive guide covering the main topics related to survey research in software engineering, it is also crafted with flexibility in mind. We hope that this facilitates the adoption of the content, methodologies, and resources provided by educators to fit their teaching styles and the specific needs of their students.

The remainder of this chapter is organized as follows. Section 2 presents a what we believe to be a typical syllabus for teaching survey research. Section 3 provides advice on how to teach the main topics of survey research, with subsections dedicated to each specific learning objective of the syllabus. Section 4 contains the concluding remarks.

2 Survey Research Course Syllabus

This section aims to offer teachers a comprehensive overview of the typical structure, content, and expectations of a course on survey research. We begin with the course description, laying the foundation for what to expect throughout the course. Thereafter, we describe the learning objectives, outlining the skills and knowledge that students are expected to acquire. We then present an example of a schedule planned to deliver a structured and effective learning experience, including assignments to reinforce learning and gauge the students' understanding of the course material. Finally, we discuss assessment possibilities and report on resources that we provide for teachers.

Course Description The course provides a comprehensive overview of survey research principles and practices. Students will be taught how to design and

Table 1 Learning objectives and Bloom's taxonomy levels

ID	Learning objective	Students will be able to ...	Bloom's taxonomy
LO1	Understanding the characteristics and purposes of survey research	... articulate on the characteristics and purposes of survey research. ... provide survey research application examples.	Remembering & Understanding
LO2	Designing and evaluating survey instruments	... create survey instruments aligning with specific research objectives and theories. ... critically assess the effectiveness of survey instruments	Evaluating & Creating
LO3	Mastering sampling and data collection	... apply best practices in sampling and data collection. ... understand the trade-offs of different sampling and data collection methods.	Understanding & Applying
LO4	Applying statistical and qualitative analysis methods	... utilize statistical and qualitative analysis techniques to interpret survey data.	Applying & Analyzing
LO5	Identifying and addressing validity and reliability threats	... analyze and address potential threats to the validity and reliability of survey research.	Analyzing & Evaluating
LO6	Understanding ethical considerations in survey research	... identify, understand, and apply ethical considerations in survey research.	Understanding & Applying

evaluate survey instruments, focusing on aligning them with research objectives and relevant theories. The course covers best practices in sampling, data collection, and both statistical and qualitative analysis techniques. A critical course component is teaching students to identify and address potential threats to the validity and reliability of survey research. Furthermore, it encompasses ethical considerations pertinent to the field of survey research.

Learning Objectives The course structure has been developed to address the learning objectives outlined in Table 1. We also indicate the levels of learning associated with each objective, as categorized by the revised version of Bloom's taxonomy [8].

Schedule Our experiences in teaching survey research range from brief and informative formats, like 2-hour seminars or 4-hour conference tutorials, to more comprehensive, in-depth treatments as part of doctorate-level programs. Given the importance of the subject and its evolving nature in software engineering, a

more extensive format that includes practical assignments and detailed discussions for each learning objective is advisable. For illustrative purposes, we present the schedule of a recent experience of the first author teaching an 18-hour doctorate-level discipline on the topic at *Universidad de Castilla-La Mancha* (UCLM) in Spain in Table 2. This particular discipline was structured into 12 classes, each lasting 1.5 hours.

It is important to note that this 18-hour schedule assumes that students will independently acquire more detailed knowledge on certain topics, such as statistical and qualitative analysis, outside the course's scope. Without this prerequisite, the course could extend far beyond 18 hours, including supplementary exercises and hands-on practice for statistical and qualitative analyses based on open survey datasets.

Assignments Considering the example course schedule, students are expected to participate actively in each session's assignments. These assignments should ideally be prepared in interactive formats to foster group discussions, e.g., using previously prepared collaborative whiteboard platforms such as Miro[1] or Mural.[2] Furthermore, besides participating in the sessions, throughout the course, students are expected to apply what they have learned to develop a survey protocol for a topic of their choice to be delivered at the end of the course. This protocol should include details on the survey goal, survey design, link to the underlying theory and related work, intended instrument evaluation, sampling and data collection strategies, statistical and qualitative analysis methods to be applied, threats to validity and mitigation actions, and ethical considerations. Depending on the number of students, this might have to be a group assignment, as students are expected to present an overview of their protocols at the end of the course.

Assessment The assessment method can certainly be adapted by the teacher. However, we recommend allocating 50% of the final grade to participation in the class assignments and the remaining 50% to the quality of the final survey protocol delivered by the students.

3 Advice on How to Teach Survey Research

We organize the advice on important aspects to cover when teaching lectures related to survey research based on the learning objectives detailed in the previous section. Recommendations are based on lessons learned from teaching and from experiences conducting large-scale international surveys.

[1] http://www.miro.com/.

[2] http://mural.co/.

Table 2 Course lectures and assignments

Class	LO	Lecture	Assignment
1	LO1	Characteristics and Purposes of Survey Research	Provide three application examples of survey research in software engineering
2	LO2	Survey Design: Basics of Survey Design	Design basic demographic questions to characterize software professionals
3	LO2	Survey Design: Goal-Question-Metric-Based Design	Break down examples of GQM goal definitions into questions, metrics, and survey questions
4	LO2	Survey Design: Theory-Driven Design	Discuss example papers using theory-driven survey design, identifying hypotheses, underlying theory, constructs, and validated scales
5	LO2	Survey Design: Survey Instrument Evaluation	Discuss survey instrument evaluation methods
6	LO3	Sampling and Data Collection	Provide examples for target populations, units of analysis, sampling strategies, invitation formats, and benchmarks for representativeness
7	LO4	Statistical Analysis: Descriptive Statistics	Apply descriptive statistics on demographic questions from a collected sample using open survey data, and argue for sample representativeness
8	LO4	Statistical Analysis: Inferential Statistics	Discuss examples of published surveys applying inferential statistics, including hypothesis testing, bootstrapping with confidence intervals, Bayesian analysis, and structural equation modeling
9	LO4	Qualitative Analysis	Discuss examples of published surveys applying grounded theory procedures and thematic analysis
10	LO5	Threats to Validity and Reliability	Discuss examples of threats to validity and reliability and mitigation actions of published surveys
11	LO6	Ethical Considerations	Draft an informed consent form and discuss ethical concerns and how to report ethical considerations properly
12	LO1–LO6	Final Survey Protocol Presentations	Present the final survey protocol

3.1 Teaching Characteristics and Purpose of Survey Research (LO1)

In our example schedule (cf. Table 2), this learning objective is taught in a single lecture. Hereafter, we provide some recommendations for teaching the characteristics and purposes during this lecture.

3.1.1 Characteristics of Survey Research

When teaching the characteristics of survey research, it is important to cover the following aspects to ensure students grasp the potential but also the limitations of this method.

Prevalence of Surveys Highlight how surveys are widely used across various fields, emphasizing their utility in capturing information from large groups of individuals [16, 46]. It is advisable to cite examples of relevant software engineering surveys and their findings to motivate the students. Some well-known examples follow. In Brazil, the *iMPS* survey, conducted 6 years in a row with hundreds of software organizations [24, 44], allowed the assessment of the impact of adopting a software process reference model on performance results [23]. The *Naming the Pain in Requirements Engineering (NaPiRE)* family of surveys revealed the requirements engineering status quo [45] and contemporary problems [15] in conventional and in machine-learning contexts [2, 22]. The *HELENA* survey transferred the principles underlying the NaPiRE survey to provide evidence on the prevalence of hybrid development approaches and allowed a reinterpretation of agility 20 years after the agile manifesto [26].The *Pandemic Programming* survey revealed how COVID-19 affected the well-being and productivity of software developers [32]. Surveys have been particularly useful in assessing human aspects related to software engineering. For instance, the survey by Graziotin et al. [18] assessed the consequences of happiness and unhappiness among software developers. Another international survey on the impostor phenomenon revealed a high prevalence of impostor feelings among software engineers, especially women [19]. Providing examples of relevant scientific findings will help motivate your students.

Timing of Surveys Discuss the timing of survey administration relative to the occurrence of the phenomena of interest. Explain how surveys can be retrospective (looking back at something that has already happened) or prospective (looking ahead to something that is expected to happen). Remind the students that in a survey, they can either ask for specific facts that participants experienced or for opinions on topics, but that, in general, from a scientific perspective, facts are preferred over opinions [42].

Lack of Experimental Control Clarify that, unlike controlled experiments, surveys do not allow for control over variables or direct manipulation of the environ-

ment. Emphasize the observational nature of survey research, which often leads to challenges in establishing causality.

Focus on Understanding Emphasize the importance of designing surveys to maximize understanding from a minimal set of variables. Teach students about the trade-offs between the breadth of information collected, the depth of understanding, and the importance of focused research questions.

Generalization to the Population Ensure that students understand that surveys are rarely conducted to create an understanding concerning a particular sample; the typical focus is on generalizing results to the population from which the sample was drawn. Surveys that conclude only on a given sample based on descriptive statistics, without applying inferential statistics for generalization purposes (or at least generalization by analogy in relation to existing evidence), are, from a scientific point of view, not as powerful as they could be for helping to building and evaluating theories, i.e., they would barely contribute to scientific progress.

3.1.2 Purpose of Survey Research

When teaching the purposes of survey research, it is essential to cover the distinct general objectives that surveys can fulfill [46, 47]. We provide examples to explain these objectives precisely.

Exploratory Surveys Emphasize the exploratory nature of these surveys as a preliminary step in research. Discuss how exploratory surveys can help identify key issues and constructs that should not be overlooked in subsequent in-depth investigations (e.g., constructs of a theory, like requirements elicitation techniques [45]). Provide examples of how these surveys can uncover new areas of interest, such as emerging techniques in a field like requirements elicitation. Such surveys may also serve the purpose of preparing for a larger investigation where the identified constructs can further be investigated by complementary research methods.

Descriptive Surveys Explain how descriptive surveys depict the characteristics of a population or phenomena. Teach the importance of clear and focused survey questions to accurately capture things like the distribution of attributes within a population (e.g., usage of requirements elicitation techniques [45]). Remember to mention the importance of analyzing and interpreting the collected data beyond the sample to make assertions about the population (e.g., using inferential statistics), such as the prevalence of certain practices.

Explanatory Surveys Define explanatory surveys and their goal to establish cause-and-effect relationships. Discuss how these surveys can provide insights into why certain phenomena occur, such as the reasons behind using specific techniques in certain contexts. For example, why are specific requirements elicitation techniques used in specific contexts [45]. Highlight the role of designing surveys with questions that can pave the way to explanatory findings, coupled with the proper utilization

of statistical techniques for data analysis; this will help motivate students for these future learning topics.

3.1.3 Assignments

After teaching the characteristics and purposes of survey research, a recommended assignment involves conducting a brainstorming session, asking each student to think out of the box about three application examples of survey research in software engineering. A collaborative whiteboard platform, such as Miro or Mural, could be used for this purpose, and the individual contributions of the students could be collectively grouped by affinity by the instructor. This assignment could also break the ice and organize the students into topic affinity groups to work together on the final survey protocol throughout the course.

Key Takeaways on Teaching Characteristics and Purposes

- Precisely explain the characteristics of survey research to ensure students understand the strengths and limitations of the method.
- Cover the distinct general objectives that surveys can fulfill, providing examples to ensure students understand the method's investigation possibilities.

3.2 Teaching Designing and Evaluating Survey Instruments (LO2)

As outlined in our example schedule (cf. Table 2), this learning objective typically requires several lectures. We recommend at least one lecture on the basics of survey design, two additional ones on specific and effective design approaches that we have been teaching, and a last one on survey instrument evaluation methods. We abstracted and conceptualized the two specific design approaches based on best practices observed in several surveys published in top-tier software engineering journals and herein introduce them to the community as *goal-question-metric-driven (GQM-driven) survey design* and *Theory-Driven Survey Design*. Recommendations for these lectures follow.

3.2.1 Basics of Survey Design

This lecture is a prerequisite for the subsequent ones, which cover specific survey design approaches. When teaching the basics of survey design, it is important to

provide a structured approach that encompasses various types of questionnaires, question types, and question categories, as well as measurement scales and conditions for obtaining accurate responses. Incorporating these elements into teaching will help students understand the fundamental principles of survey design and develop the skills necessary to create effective and reliable survey instruments. Recommendations for teaching them follow.

Questionnaire Types Explain the difference between self-administered and interviewer-administered questionnaires, including their advantages and contexts of use. Discuss the importance of considering the respondent's environment and potential distractions for self-administered questionnaires. For interviewer-administered questionnaires, highlight interviewer effects and the need for standardization in question delivery.

Question Types Demonstrate how to construct open-ended, close-ended, and hybrid questions, including when and why to use each type. Practice writing questions with the class and analyze examples for clarity and bias.

Question Categories Define demographic, substantive, filter, and sensitive questions, providing advice on how to integrate them into a survey. Discuss ethical considerations when asking sensitive questions and the use of filter questions to guide respondents through the survey.

Measurement Scales Teach the characteristics of nominal, ordinal, interval, and ratio scales, and provide examples of each. Use exercises to help students choose the appropriate scale for different types of data they want to collect.

Conditions for Appropriate Responses Emphasize the need for questions to be understandable, ensuring they match the target population's language proficiency and cultural context. Ensure that respondents of that population (and your sample) have the knowledge to answer the questions accurately and discuss strategies to encourage their motivation and willingness to participate.

Common Question Wording Problems Offer strategies to avoid common question-writing pitfalls such as using complex language, technical terms, double-barreled questions, and double negatives. Highlight the importance of keeping questions concise to prevent misunderstanding and respondent fatigue.

Opinions vs. Facts Differentiate between questions that seek subjective opinions and those that ask for more factual phenomena (observed or experienced by respondents). Discuss the relevance of each in relation to the survey's goals and the implications for data analysis.

Assignments A recommended assignment after teaching the basics of survey design involves asking students to design basic demographic questions, for instance, to characterize software professionals. Engage students to review their questions critically and to compare them to questions contained in well-known surveys that also involve software professionals, such as the Stack Overflow Annual Developer survey [31]. Explain to them, however, that even the demographics to be collected

strongly depend on the goal of the survey. Better aligning survey questions to research goals is the topic of the next suggested lecture.

3.2.2 Goal-Question-Metric-Driven Survey Design

This lecture focuses on teaching a specific survey design approach that helps design concise surveys focused on relevant goals and research questions. Although we are conceptualizing it as part of this book chapter, we have been teaching it for years, and it has been explicitly or implicitly applied in several software engineering surveys that carefully describe the rationale of deriving questions and metrics from a clear research goal (e.g., [2, 7, 19, 23, 24, 26, 29]). Advice on how to teach the approach and assignments follows.

The *Goal-Question-Metric-Driven Survey Design* approach adapts the more generic Goal-Question-Metric (GQM) measurement approach [5, 6] to survey design. GQM defines a generic way to plan and execute measurement and analysis activities. It starts with the declaration of the measurement *Goals*. From the goals, *Questions* that we would like to answer with the data interpretation are defined. Finally, from the questions, the *Metrics* and the data to be collected are defined. The adaptation for survey design consists of using the GQM goal definition template to precisely describe the purpose of the survey, generating research questions to be answered from the goal definition, and then generating metrics collected to answer these questions. The survey instrument is then designed following the basic survey design principles with questions to collect these metrics objectively.

We recommend teaching the GQM-driven survey design approach by providing examples of precise goals defined by applying the GQM goal definition template to surveys. The GQM goal definition template is defined as follows:

```
Analyze <object of study>
with the purpose of <goal>
with respect to <quality focus>
from the point of view of the <perspective>
in the context of <context>.
```

We recommend to pay special attention when teaching to fill this template for surveys correctly. The object of study corresponds to the phenomena of interest to be investigated. The goal is typically characterizing (for explorative or descriptive surveys) or understanding (for explanatory surveys). The quality focus will be directly related to the metrics to be collected through the survey's substantive questions. The perspective should reflect the survey population (in the case of more direct questions) or the researcher (in the case of using more robust validated scales to be interpreted by the researcher). Finally, the context allows further detailing of the data collection context.

An example goal definition, taken from the iMPS survey [24], which assessed the performance of software organizations that adopted the Brazilian software process reference model (MPS-SW [35]), follows. "Analyze *the profile of software development organizations* with the purpose of *characterizing* with respect to

the organizations' current profile, satisfaction degree regarding the MPS model, variation of presence in international markets, variation of exportation volume, and variation concerning cost, estimation accuracy, productivity, quality, user satisfaction, and return of investment (ROI) from the point of view of *software development organizations* in the context of *software development organizations with unexpired MPS-SW assessments published in the SOFTEX portal*."

Following the GQM-driven survey design approach, this goal was detailed into questions, which were then detailed into metrics [44]. The survey was then carefully designed to capture these metrics. To exemplify this process, estimation accuracy is one of the elements of the quality focus. To capture this quality focus element, the question was, "What is the organizations' estimation accuracy?" However, this is a research question and not yet a survey question. First, we have to break it down into metrics. The metrics that we were looking for were the *average project duration* and the *average estimated project duration*. Based on these base metrics, we could then calculate the estimation accuracy. Hence, the estimation accuracy element led us to design two survey questions: "What is the average duration of projects conducted within the last 12 months, measured in months?" and "What is the average estimated duration of projects conducted within the last 12 months, measured in months?"

This example leads us to an important aspect that should be emphasized when teaching this approach. The GQM questions do not necessarily directly relate to the survey questions. Instead, the survey questions should be designed to focus on the metrics (and scales) and the data to be collected. There are several examples of published surveys explicitly using the GQM goal definition template and the question and metrics breakdown rationale to support designing the survey (e.g., [7, 13, 19, 29]).

Assignments A recommended assignment after teaching GQM-driven survey design involves providing students with examples of GQM goal definitions taken from research papers, asking students to break them down into questions and metrics. Thereafter, they should be asked to design survey questions to collect these metrics from population sample. Using the GQM goal definition template from the survey by Guenes et al., which concerns the impostor phenomenon, could allow an interesting reflection linking to the next lecture, as it will break down to metrics that correspond to theoretical constructs (e.g., impostor phenomena score or perceived productivity) that will require more complex approaches, such as validated scales and careful theoretical construct breakdowns, for proper measurement.

3.2.3 Theory-Driven Survey Design

This lecture focuses on teaching a more robust survey design approach that helps design surveys to support theory building and evaluation. A theory provides explanations and understanding in terms of basic constructs and underlying mechanisms. From a practical perspective, theories are useful and explain or predict phenomena

that occur in software engineering; from a scientific perspective, theories should guide and support further research in software engineering [21].

We conjecture that the type of research to be supported by theories includes survey research. In fact, theory building and evaluation, as elaborated in the introductory editorial chapter to the edited book this chapter is part of, can guide the design and analysis of surveys, and surveys can also be applied to evaluate theories [46]. Although we are conceptualizing the theory-driven survey design approach as part of this chapter, we have been teaching it for years; theory has guided the design of several robust software engineering surveys (e.g., [18, 19, 32, 45]). Advice on how to teach the approach and assignments follows.

Theory building and survey research can be strongly interrelated. The *theory-driven survey design* approach relies on drafting a theory based on observations and available literature and designing the survey based on that theory to test or potentially extend it. According to Sjoberg et al. [37], the main building blocks of such a theory are constructs, propositions, explanations, and scope. Constructs describe what the basic elements are, propositions how the constructs interact with each other, explanations why the propositions are as specified, and the scope elaborates what the universe of discourse is in which the theory is applicable.

An initial theory may be as simple as a taxonomy of constructs or a set of statements relating to constructs. For the NaPiRE [45] and the Impostor Phenomenon [19] surveys, a set of constructs and propositions (hypotheses to be tested) were elaborated based on available literature and expert knowledge. For the Pandemic Programming survey [32], a multifactorial theoretical model relating a set of constructs was designed based on related work. In both cases, the surveys were designed to test the theory using inferential statistics. We will use the theoretical model of the pandemic programming survey to highlight some key issues to be emphasized when teaching theory-driven survey design.

The theoretical model of developer well-being and productivity, which supported the design of the Pandemic Programming survey, is shown in Fig. 1. The survey assessed ten hypotheses revolving around the five constructs shown in the model. H1 and H2 stated that working from home due to COVID-19 would reduce well-being and perceived productivity, respectively. The arrows depict the remaining eight hypotheses, and more details can be obtained in the paper [32]. It is important

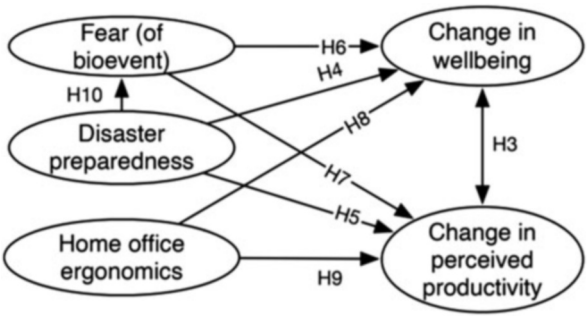


Fig. 1 Exemplary theoretical model of the developer well-being and productivity survey [32]

to highlight that the constructs and the hypotheses should not be randomly drawn. Someone applying theory-driven survey design should do their homework, looking for related work in the literature to identify theoretical constructs and hypothesize meaningful relations, explaining the rationale behind such relations. For instance, the rationale for H1 is based on related work indicating that uncertainty and isolation induce feelings of frustration, anxiety, and fear [12, 39, 40]. Therefore, it would be likely for developers to experience reduced emotional well-being.

Another relevant aspect of the theory-driven survey design approach that should be highlighted when teaching the subject is to use validated scales as part of the survey to measure the constructs, as much as possible, to improve the survey's construct validity. For instance, the pandemic programming survey used the World Health Organization's five-item well-being index (WHO-5) to assess emotional well-being. The WHO-5 scale is widely used, widely applicable, and has high sensitivity and construct validity [41]. Similarly, the Impostor Phenomenon survey used the Clance Impostor Phenomenon Scale (CIPS) to measure how often and seriously the impostor phenomenon interferes with a person's life. CIPS is internationally validated and is the most used scale by researchers and practitioners [28].

In summary, survey research and theory building are strongly interrelated. The exact relationship depends on whether the theory is descriptive, explanatory, or predictive. Hence, theories are of high value to guide the design of robust surveys. On the other hand, survey data supports the definition or refinement of constructs, relationships, explanations, and the scope of a theory, as well as the testing of a theory. Validated scales should be used as much as possible to improve construct validity. So far, the software engineering community still falls short in maturity concerning building validated scales, which is an inherently complex task that has, however, been successfully achieved in other fields, such as psychology [14]. Building (and using) robust scales could help take the maturity of survey research in software engineering to the next level. These discussions should be part of this lecture.

Assignments A recommended assignment after teaching theory-driven survey design involves providing students with examples of research papers using the theory-driven approach (e.g., [19, 32, 45]), asking the students to identify theoretical constructs, related scales, hypotheses (or propositions), and underlying theory. As part of our complimentary material, we provide a Miro board where students can collectively work on a research paper registering the demographics and related benchmarks, the constructs and related scales, and the hypotheses (or propositions) and their related underlying theories.

3.2.4 Survey Instrument Evaluation

This final lecture on survey design focuses on teaching instrument evaluation methods. Several survey instrument evaluation methods can be used to assess the validity and reliability of the survey instrument. We recommend covering the fol-

lowing methods, which are typically related to this purpose [27, 34]: expert reviews, focus groups, pilot surveys, cognitive interviews, and experiments. Additionally, the empirically evaluated checklist for surveys in software engineering by Molléri et al. [30] can be used as an additional valuable resource for evaluating the survey design (as well as the final survey report).

Encourage students to understand the role of expert reviews in assessing content validity. Discuss how to select experts and what criteria they should use to evaluate the survey questions. With respect to focus groups, explain how they can be utilized to gather feedback on the survey's comprehensibility, relevance, and overall design. Highlight the importance of diverse participant selection to ensure a wide range of perspectives. Pilot surveys are highly recommended. Teach the purpose and process of conducting pilot surveys, emphasizing their role in identifying issues with survey instructions, question-wording, and format. Discuss how to analyze pilot survey data to make necessary adjustments before full deployment. Remember to describe cognitive interviewing techniques for evaluating how respondents perceive and process survey questions. You may practice mock cognitive interviews in class to give students a hands-on experience. Finally, introduce the concept of using experimental designs to test different aspects of the survey instrument, such as question order effects. Discuss how to set up control and experimental groups and measure the impact of changes to the survey.

Assignments The assignment for this lecture may involve discussing the different survey instrument evaluation methods and how they were applied in surveys published in the software engineering realm.

Key Takeaways on Teaching Designing and Evaluating Survey Instruments

- Do not forget the basics! Teach the "when" and "why" of different types of questionnaires, question types, and question categories, as well as measurement scales and conditions for obtaining accurate responses.
- Teach GQM-driven and theory-driven survey design approaches to help students design focused and concise surveys, maximizing understanding based on a minimal set of variables related to relevant research goals or theories.
- Emphasize the importance of using validated scales as much as possible to improve construct validity.
- Emphasize that survey instruments should be evaluated using different methods to avoid threats to validity and improve reliability.

3.3 Teaching Sampling and Data Collection (LO3)

The third learning objective covers sampling and data collection. In our example schedule (cf. Table 2), it is taught in a single lecture. However, when having an entire course dedicated to survey research, there would surely be space to devote more attention to this important topic. Hereafter, we provide some recommendations during this lecture when teaching each of these topics.

3.3.1 Sampling

Baltes and Ralph [4] critically reviewed sampling in software engineering research and found that sampling, representativeness, and randomness often appear misunderstood, highlighting the importance of properly teaching these aspects. Hence, when teaching survey research sampling, it is critical to define the target population precisely and impart the concepts of representativeness and sample size. Different sampling strategies and their intricacies should also be taught. We provide recommendations on these elements hereafter.

Defining the Target Population Start by discussing the importance of clearly identifying the target population the survey aims to generalize to. Use real-world examples to illustrate how the unit of analysis can vary in software engineering surveys [11], such as organizations, software teams or projects, and individuals. Stress the importance of clearly defining the target population to ensure the survey's relevance and applicability (and reproducibility).

Representativeness Explain the concept of representativeness and its significance in generalizing survey results to the broader population. Highlight the challenges in software engineering research due to the lack of comprehensive demographic information from traditional sources like government statistical offices. Discuss alternative sources of demographic data, such as commercial data providers and community-driven surveys like the Stack Overflow Annual Developer Survey [31]. They have the bias that only people registered at Stack Overflow can be sampled. Yet this could be tolerated in light of the popularity of the platform among software developers. If possible, conduct a class activity to analyze the demographic data from these sources and discuss how they can help define the sampling strategy.

Teach students how to compare the demographics of their survey sample with larger datasets to assess representativeness. Emphasize the importance of transparency in reporting the degree of representativeness to prevent overclaiming and to enhance research credibility. An example of a survey with such a representativeness assessment visually comparing the demographics of the gathered sample against those of the Stack Overflow Annual Developer Survey can be seen in the Impostor Phenomenon survey [19].

Sample Size Estimation Introduce the concept of sample size and its critical role in statistical analysis and the conclusion validity of surveys. Explain formulas for

calculating appropriate sample sizes for a population. A simple way, for example, is to follow Yamane [48]. As described by Wagner et al. [46], he proposed to use the following equation to calculate a suitable sample size n:

$$n = \frac{N}{1 + Ne^2} \tag{1}$$

In Eq. 1, N is the population size, and e is the level of precision. For the population estimate of 23 million developers worldwide and a level of precision of 0.05, this would require a sample of size 400. Hence, explain that for most intents and purposes, a sample size of more than 400 allows to claim strong generalizability given that the representativeness was also checked, as previously described. To avoid raising expectations, inform that most surveys may fall short of this, but explain that a clear discussion comparing the sample size and representativeness with these figures makes it easy to evaluate the strengths and weaknesses of a particular survey.

Sampling Strategies Survey sampling strategies are crucial to understand because they directly impact the validity and generalizability of survey research results. Linaker et al. [27] present some common sampling strategies, dividing them into non-probabilistic and probabilistic sampling. Hereafter, we briefly introduce sampling strategies to be covered, along with some advice for teaching them.

- **Non-probabilistic sampling**. Related to all sampling approaches in which randomness could not be observed in selecting the units.
  - *Convenience (accidental) sampling.* Choosing individuals who are easily accessible or willing to participate, such as students in a classroom. Discuss the limitations of convenience sampling, such as potential bias.
  - *Quota sampling.* Ensuring that the sample includes certain percentages of groups, reflecting their proportions in the target population. Create scenarios for students to design quota samples based on given population characteristics.
  - *Purposive (judgment) sampling.* Selecting participants based on specific characteristics or qualities, making them suitable for answering the research question. Engage students in discussions about when purposive sampling is appropriate (e.g., expert surveys, such as [29]), and have them outline a plan for selecting a purposive sample.
  - *Snowball sampling.* Asking participants to refer others to the study, which is particularly useful for hard-to-reach or specialized populations. Discuss the potential implications of participant referrals.
- **Probabilistic sampling**. All units from a sampling frame must have the same probability of being selected.
  - *Simple random sampling.* Selecting individuals from a larger population where each individual has an equal probability of being chosen. Demonstrate the use of random number generators or other tools to achieve a random sample.

- *Clustered sampling.* Dividing the population into clusters (e.g., geographic areas) and randomly selecting within these clusters. Provide cases where cluster sampling is appropriate.
- *Stratified sampling.* Dividing the population into strata (e.g., age, income level), and then randomly sampling from each stratum to ensure representation across key variables. Teach how to stratify your sample based on distributions of the populations to increase sample representativeness.
- *Systematic sampling.* Choosing every nth individual from a list after a random start. Discuss that, in general, systematic random sampling is preferred but that the systematic sampling approach can also have benefits, such as assuring that the sample is spread evenly over a population, reducing the risk of accidental clustering.

3.3.2 Data Collection

There are essentially two strategies to approach the target population to collect data in questionnaire-based surveys, having very distinct implications on the survey design and the recruitment approaches [46]:

- *Closed invitations* follow the strategy of approaching known groups or individuals to participate in a survey per invitation only and restrict the survey access only to those being invited.
- *Open invitations* follow the strategy of approaching a broader, often anonymous audience via open survey access; i.e., anyone with a link to the survey can participate.

When teaching data collection in survey research, the implications of these strategies should be precisely explained to enable informed decisions. We hereafter summarize implications that should be taught, based on challenges reported by Wagner et al. [46]. Closed invitations are suitable when it is possible to precisely identify and approach a well-defined sample of the target population. They may be required when filtering out participants who are not part of the target population would be difficult, harming the sample representativeness. Open invitations, on the other hand, allow reaching out for larger samples. However, they typically require more careful consideration of context factors when designing the survey instruments. These context factors can then be used during the analyses to filter out participants that are not representative (e.g., applying the blocking principle to specific context factors), improving the survey instrument's criterion validity.

3.3.3 Assignments

As survey data collection is hard to enact in teaching scenarios, we recommend an assignment for the students to provide and discuss examples of target populations,

detailing the units of analysis, sampling strategies, invitation formats, and benchmarks for representativeness.

Key Takeaways on Teaching Sampling and Data Collection

- Teach sampling fundamentals and strategies (and their implications). Sampling, representativeness, and randomness are often misunderstood in software engineering research.
- Discuss the strategies to approach the target population and their implications to allow students to make informed decisions when planning their surveys.

3.4 Teaching Statistical and Qualitative Analysis (LO4)

The next learning objective covers statistical and qualitative analysis. We recommend lectures on descriptive and inferential statistics and an additional lecture on qualitative analysis. Of course, the depth in which these lectures are covered will depend on the students' background. Recommendations for these lectures follow.

3.4.1 Statistical Analysis

Although surveys can be qualitative, most often, the majority of the questionnaires are composed of closed questions that have quantitative results. Therefore, it is fundamental to teach the main concepts of statistical analysis and show the students their possibilities for analyzing quantitative survey results. In particular, we recommend emphasizing that the statistical analysis should compromise the sample's descriptive statistics and inferential statistics to generalize the results to the population.

Descriptive Statistics The goal of descriptive statistics is to characterize the answers to one or more questions of our specific sample. In surveys, they provide a way to summarize large samples to make them understandable at a glance. Items to be covered when teaching descriptive statistics include measures of central tendency, measures of variability, frequency distributions, and graphical representations.

Which descriptive statistic is suitable depends on what we are interested in most and the scale of the data. Teach how to choose the appropriate measures and graphical representation depending on the data type and what you want to convey.

For ordinal scales (e.g., Likert scales), we can use frequency counting, mode, median, minimum, maximum, median absolute deviation, and interquartile range.

For graphical representation, we recommend providing examples showing the distribution of ordinal data in stacked bar charts.

For interval or ratio scales, we can use all available descriptive statistics, such as mean, variance, and standard deviation. For graphical representation, we recommend providing examples using boxplots, which provide a descriptive overview and enable eliminating outliers by using the quartile method.

Inferential Statistics While descriptive statistics concern the sample, inferential statistics allow making predictions and generalizations about a population. Typically these statistics are the ones used to answer the research questions defined in the survey design. When teaching inferential statistics, we recommend explaining different possibilities for analyzing quantitative survey results, including null hypothesis significance testing, bootstrapping with confidence intervals, Bayesian analysis, and structural equation modeling.

As the name suggests, null hypothesis significance testing is particularly useful for testing hypotheses related to a theory. Propositions can be operationalized into hypotheses to test with the survey data. In surveys, we typically can have point estimate hypotheses for answers to a single question or hypotheses on correlations between answers to two questions. Several statistical significance tests can be applied, with differences in their statistical power. Of course, these tests require a representative sample. Examples of such tests include Mann-Whitney, t-test, Wilcoxon signed-rank, paired t-test, Kruskal-Wallis, ANOVA, and Tukey. The characteristics of such tests and their assumptions should be taught. Ideally, the statistical test for which the assumptions are met with the highest power should be used to evaluate the hypotheses.

One of the limitations of null hypothesis significance testing lies in its dichotomous nature. An alternative lies in using bootstrapping to calculate confidence intervals [46], which allows for avoiding fixed significance level thresholds. The idea of bootstrapping in this context is to repeatedly take samples with replacements from the survey sample and calculate the statistic we are interested in. This is repeated a large number of times and, thereby, provides us with an understanding of the distribution of the sample. An example taken from the NaPiRE survey [45] is shown in Fig. 2, where the confidence intervals allow observing that more than 60% of the organization's elicit requirements using interviews or facilitated meetings.

Another alternative comprises Bayesian analysis [43], where probability is understood as a representation of the state of knowledge or belief. This approach acknowledges uncertainty and allows the gradual integration of evidence and accumulating knowledge. It is suitable for situations in which you are building knowledge based on several survey trials.

Finally, structural equation modeling can be employed to analyze multiple variables at the same time, allowing an understanding of complex relationships between variables to validate theoretical models. The *Pandemic Programming* [32] survey represents an application example of this statistical analysis technique in the context of software engineering surveys.

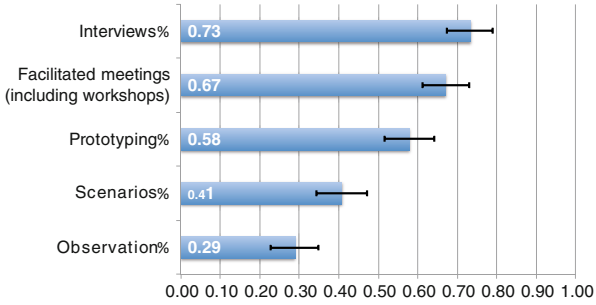


Fig. 2 Proportions with confidence intervals for question "How do you elicit requirements?" [45]

3.4.2 Qualitative Analysis

When teaching qualitative analysis in the context of survey research, emphasize the role of open questions. Open questions allow respondents to freely express their perspectives and perceptions, providing a rich dataset for analysis. Highlight that these types of questions can provide detailed insights into a phenomenon, potentially revealing causal relationships among theoretical constructs and leading to new theoretical explanations. However, clarify that analyzing the resultant large volumes of qualitative data is not trivial and requires substantial effort.

We recommend introducing thematic analysis [9] and grounded theory [17] as pivotal methods for supporting qualitative analyses of data derived from open questions. Thematic analysis allows for the identification, analysis, and reporting of themes within data, offering a flexible approach that can be either deductive or inductive, depending on the research goals. Grounded theory, on the other hand, involves inductively generating theories from data, helping researchers to systematically identify themes and build theoretical frameworks from the patterns that emerge in the qualitative responses.

It is important to emphasize that correctly applying qualitative methods requires a deep understanding of their respective procedures and nuances. For instance, grounded theory has several variants, and Stol et al. [38] provide specific considerations for properly conducting and reporting grounded theory. For further advice on teaching qualitative methods, we recommend referring to Christoph Treude's chapter on qualitative data analysis in software engineering in this edited book.

3.4.3 Assignments

Recommended assignments related to teaching statistical and qualitative analysis organized based on the class schedule in Table 2 include the following:

Descriptive statistics assignment: Students access an open survey dataset, perform descriptive statistics on demographic data, and assess the sample's representativeness by comparing these statistics to established benchmarks. Guenes et

al. [19] provide an example survey providing such an open dataset and including representativeness analysis.

Inferential statistics assignment: Students review survey research articles that utilize different inferential statistic methods, such as hypothesis testing (e.g., [10, 19, 45]), bootstrapping (e.g., [2, 19, 45]), and structural equation modeling (e.g., [20, 32]). Each student should select one article, summarize the use of inferential statistics, and critically assess how these methods helped address the research question.

Qualitative analysis assignment: Students review survey research articles that involve qualitative analysis and apply procedures from grounded theory and/or thematic analysis (e.g., [15, 22]). For each article, students will describe how the method was used, assess its effectiveness in contributing to the research goal, and identify any limitations.

It is noteworthy that, focusing on the specifics of survey research, we did not include more in-depth hands-on inferential or qualitative analysis assignments. This may, however, vary according to the specific learning objectives.

Key Takeaways on Teaching Statistical and Qualitative Analysis

- Teach descriptive statistics and alternatives for inferential statistics, showing the students their possibilities for analyzing quantitative survey results.
- Emphasize the role of open questions and teach alternatives for qualitative analysis, with explanations of their adoption rationales.

3.5 Teaching Threats to Validity and Reliability (LO5)

Validity is a property of inferences, and the focus of surveys is on reaching conclusions that involve making inferences about the population. Hence, every survey faces threats to validity. As outlined in our example schedule (cf. Table 2), a lecture dedicated to this topic could address this learning objective.

The notions of validity and reliability are central to understanding the thoroughness and trustworthiness of the survey [25] and should be taught with special attention. Validity refers to whether the questionnaire measures what it is supposed to. Reliability regards if the results are generalizable. Hereafter we provide advice on how to teach about validity and reliability in survey research.

3.5.1 Validity

In psychometrics, the validity of a survey instrument (test) concerns "the degree to which evidence and theory support the interpretation of test scores for proposed

uses of tests" [1]. However, we mainly lack such mature instruments in software engineering. While we should rely on validated instruments as much as possible, software engineering survey instruments are typically designed in a customized way based on specific research goals and explorative, descriptive, or explanatory purposes. Therefore, we typically aim to assess validity more informally, discussing whether it is possible to safely conclude that a survey measures what it is supposed to measure. We recommend teaching and discussing the following four types of validity for software engineering survey instruments:

- *Face validity.* Focuses on the appearance of the survey and its instrument and refers to how suitable the content of a test appears to be on the surface. Mitigating face validity threats typically involves a lightweight review of the questionnaire by randomly chosen respondents.
- *Content validity.* Concerns comprehensive coverage of the subject matter. Mitigating content validity threats typically involves a group of reviewers evaluating the questionnaire. The group should include subject matter experts to ensure accurate usage of content-related concepts and respondents from the target population to ensure that the terms used for these concepts are easily understandable to that population.
- *Criterion validity.* Measures how effectively a survey predicts outcomes or correlates with other established benchmarks. It indicates if a survey works well in practical situations. Mitigating criterion validity issues in software engineering surveys typically involves assuring that the survey can accurately separate respondents from the target population and comparing the results to establish benchmarks for that population.
- *Construct validity.* Concerns theoretical soundness of the survey. It assesses how well the questions actually measure the constructs they were intended to measure. A way to mitigate construct validity concerns is to rely on validated instruments for the constructs involved in the survey as much as possible and to clearly include the rationale behind any needed adaptation.

3.5.2 Reliability

Teaching reliability (aka external validity or generalizability) in survey research involves educating students on how to ensure that their surveys produce dependable results that can be properly generalized for the population [33].

We recommend teaching and discussing reliability concerns that are relevant to survey research in software engineering, which include:

- *Internal consistency.* Measures whether different items on the same survey produce consistent results. In software engineering surveys, this could involve triangulating answers to related questions (e.g., productivity, size, and effort) or even require adding some extra questions for validation purposes.
- *Test-retest reliability.* Assesses the consistency of results when the same survey is administered to the same group of people under similar conditions at different

points in time. According to Kitchenham and Pfleeger [25], if the correlation between both of the answers is greater than 0.7, the test-retest reliability can be considered good.

- *Inter-observer reliability.* Ensures that the data collected in surveys or studies are consistent across different observers, enhancing the credibility and generalizability of the results. This is particularly important to mitigate interview bias in not self-administered surveys and to mitigate issues in qualitative analyses involving analyzing open-ended questions. It is typically addressed by having two or more observers involved in the interview and analysis process. For the analysis, it is common to measure the agreement between these raters, for instance, using Cohen's Kappa (for two raters) or Fleiss' Kappa (for three or more raters). Actions to improve the analysis consistency may involve having all raters assess a few cases and then discuss their ratings, which can help identify discrepancies and refine the rating process.
- *Statistical generalizability.* Concerns ensuring that the results of a survey can be applied to a broader population beyond the specific sample studied. This is directly related to the sample representativeness and size, as well as to the inferential statistics applied to draw conclusions. This is a good point to remind the students of the importance of using robust sampling strategies and accurate inferential statistics. The sample's representativeness should be analyzed by comparing it against established benchmarks (when those exist). If the sample is representative, then inferential statistics can be used to draw conclusions (e.g., testing hypotheses or calculating confidence intervals).

3.5.3 Assignments

A practical student assignment related to threats to validity and reliability could involve a critical analysis of existing published surveys. Students select and review specific surveys from software engineering literature, identify potential threats to both validity and reliability, and discuss how these threats could affect the conclusions of the survey. Furthermore, they could propose and discuss possible mitigation actions for the identified threats.

Key Takeaways on Teaching Threats to Validity and Reliability

- Teach the concepts of validity and reliability with special attention. These are central to understanding the thoroughness and trustworthiness of software engineering surveys.
- Teach and discuss the different types of validity relevant to survey research in software engineering.
- Teach and discuss reliability concerns that are relevant to survey research in software engineering.

3.6 Teaching Ethical Considerations

Ethical considerations are paramount in survey research within software engineering, as they ensure ethical involvement of participants and ethical treatment of their data, the integrity of the data, and the credibility of the research findings. Educating students about these ethical concerns is essential for fostering responsible research practices. We need to emphasize the importance of considering thoughtfully how and whom we contact for a survey study [46].

While ethical issues have been discussed in empirical software engineering [36] and while there are well-established regulations that touch upon selected aspects of ethics (such as privacy preservation by General Data Protection Regulation—GPPR), there are still currently no established standards or guidelines on conducting surveys ethically in software engineering. However, general principles for survey research ethics from other areas [16, 33] can be directly applied to our context. These include following informed consent rules and respecting confidentiality and privacy.

As for the informed consent, participants must be fully informed about the nature of the research, what it involves, the risks and benefits, and their rights to withdraw at any time without penalty. With respect to privacy and confidentiality, researchers must protect the privacy of participants and the confidentiality of their data, using data encryption and anonymization techniques where appropriate. Students should also be taught how to report ethics considerations in software engineering publications [3].

Another important aspect is teaching students the importance of submitting survey research to institutional ethics review boards, as they will ensure the research adheres to ethical standards and protects participants. This process also educates students on the complexities of ethical research. In professional settings, corporate ethics committees perform a comparable role.

As shown in Table 2, an assignment for this lecture could involve students drafting an informed consent form for a software engineering survey based on what should typically be included in such document [3] and discussing ethical concerns related to risks and benefits and potential implications.

Key Takeaways on Teaching Ethical Considerations

- Emphasize that ethics needs to be considered before contacting potential survey participants.
- Explain that participants must be fully informed about the nature of the research, what it involves, the risks and benefits, and their rights to withdraw at any time without penalty.
- Explain the role of the institutional ethics review boards and how to report survey ethics in software engineering publications.

4 Concluding Remarks

Throughout this chapter, we have explored effective strategies for teaching survey research, combining theoretical foundations with practical applications. From detailing a course syllabus to offering actionable advice based on seasoned experiences, the presented guidance aims to support educators in fostering a thorough understanding of survey research among their students. Additionally, we provide online resources, including slides for an entire course dedicated to teaching survey research.

We hope that the advice and materials provided in this chapter will help educators in setting up own courses or reflect upon their already existing courses when teaching survey research in software engineering. We emphasize that teaching material and advice will never be final and that educators are encouraged to adapt these materials as they see fit, customizing them to better suit their unique teaching styles and meet the specific needs of their students.

Online Material

In our online material repository https://zenodo.org/doi/10.5281/zenodo.11544897, the reader can find different slide decks that introduce the topic area of teaching survey research in different educational settings and including units of different lengths.

References

1. AERA, APA, NCME: Standards for Educational and Psychological Testing. American Educational Research Association, Washington, DC (2014)
2. Alves, A.P.S., Kalinowski, M., Giray, G., Mendez, D., Lavesson, N., Azevedo, K., Villamizar, H., Escovedo, T., Lopes, H., Biffl, S., et al.: Status quo and problems of requirements engineering for machine learning: Results from an international survey. In: International Conference on Product-Focused Software Process Improvement, pp. 159–174. Springer (2023)
3. Badampudi, D.: Reporting ethics considerations in software engineering publications. In: 2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), pp. 205–210. IEEE (2017)
4. Baltes, S., Ralph, P.: Sampling in software engineering research: A critical review and guidelines. Empir. Software Eng. **27**(4), 94 (2022)
5. Basili, V.R., Caldiera, G., Rombach, H.D.: Goal question metric paradigm. Encyclopedia Software Eng. **1**(528–532), 6 (1994)
6. Basili, V.R., Rombach, H.D.: The tame project: Towards improvement-oriented software environments. IEEE Trans. Software Eng. **14**(6), 758–773 (1988)
7. Batista, M., Magdaleno, A., Kalinowski, M.: A survey on the use of social bpm in practice in Brazilian organizations. In: Anais do XIII Simpósio Brasileiro de Sistemas de Informação, pp. 436–443. SBC (2017)
8. Bloom, B.S., Krathwohl, D.R.: Taxonomy of Educational Objectives: The Classification of Educational Goals. Book 1, Cognitive Domain. Longman (2020)
9. Braun, V., Clarke, V.: Using thematic analysis in psychology. Qual. Res. Psychol. **3**(2), 77–101 (2006)

10. Cabral, R., Kalinowski, M., Baldassarre, M.T., Villamizar, H., Escovedo, T., Lopes, H.: Investigating the impact of solid design principles on machine learning code understanding. In: 2024 IEEE/ACM 3rd International Conference on AI Engineering–Software Engineering for AI (CAIN), pp. 1–11. ACM (2024)
11. de Mello, R.M., Da Silva, P.C., Travassos, G.H.: Investigating probabilistic sampling approaches for large-scale surveys in software engineering. J. Software Eng. Res. Develop. **3**(1), 1–26 (2015)
12. DiGiovanni, C., Conley, J., Chiu, D., Zaborski, J.: Factors influencing compliance with quarantine in Toronto during the 2003 SARS outbreak. Biosecur. Bioterrorism Biodefense Strat. Practice Sci. **2**(4), 265–272 (2004)
13. Falcão, R., Villela, K., Vieira, V., Trapp, M., de Faria, I.L.: The practical role of context modeling in the elicitation of context-aware functionalities: a survey. In: 2021 IEEE 29th International Requirements Engineering Conference (RE), pp. 35–45. IEEE (2021)
14. Felipe, D.A., Kalinowski, M., Graziotin, D., Natividade, J.C.: Psychometric instruments in software engineering research on personality: Status quo after fifty years. J. Syst. Software **203**, 111740 (2023)
15. Fernández, D.M., Wagner, S., Kalinowski, M., Felderer, M., Mafra, P., Vetrò, A., Conte, T., Christiansson, M.T., Greer, D., Lassenius, C., et al.: Naming the pain in requirements engineering: Contemporary problems, causes, and effects in practice. Empir. Software Eng. **22**, 2298–2338 (2017)
16. Fowler Jr, F.J.: Survey Research Methods. Sage Publications (2013)
17. Glaser, B., Strauss, A.: The Discovery of Grounded Theory: Strategies for Qualitative Research. Routledge, New York (1967)
18. Graziotin, D., Fagerholm, F., Wang, X., Abrahamsson, P.: What happens when software developers are (un) happy. J. Syst. Software **140**, 32–47 (2018)
19. Guenes, P., Tomaz, R., Kalinowski, M., Baldassarre, M.T., Storey, M.A.: Impostor phenomenon in software engineers. In: 2024 IEEE/ACM 46th International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS) (2024)
20. Gupta, V., Kapur, P.K., Kumar, D.: Modeling and measuring attributes influencing DevOps implementation in an enterprise using structural equation modeling. Inf. Software Technol. **92**, 75–91 (2017)
21. Hannay, J.E., Sjoberg, D.I., Dyba, T.: A systematic review of theory use in software engineering experiments. IEEE Trans. Software Eng. **33**(2), 87–107 (2007)
22. Kalinowski, M., Mendez Daniel Giray, G., Alves, A.P.S., Azevedo, K., et al.: Naming the pain in machine learning-enabled systems engineering. arXiv:2406.04359 (2024)
23. Kalinowski, M., Weber, K., Franco, N., Barroso, E., Duarte, V., Zanetti, D., Santos, G.: Results of 10 years of software process improvement in Brazil based on the MPS-SW model. In: 2014 9th International Conference on the Quality of Information and Communications Technology, pp. 28–37. IEEE (2014)
24. Kalinowski, M., Weber, K.C., Travassos, G.H.: imps: an experimentation based investigation of a nationwide software development reference model. In: Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, pp. 336–338 (2008)
25. Kitchenham, B.A., Pfleeger, S.L.: Personal opinion surveys. In: Guide to Advanced Empirical Software Engineering, pp. 63–92. Springer (2008)
26. Kuhrmann, M., Tell, P., Hebig, R., Klünder, J., Münch, J., Linssen, O., Pfahl, D., Felderer, M., Prause, C.R., MacDonell, S.G., et al.: What makes agile software development agile? IEEE Trans. Software Eng. **48**(9), 3523–3539 (2021)
27. Linaker, J., Sulaman, S.M., Höst, M., de Mello, R.M.: Guidelines for Csurveys in Software Engineering v. 1.1, vol. 50. Lund University (2015)
28. Mak, K.K., Kleitman, S., Abbott, M.J.: Impostor phenomenon measurement scales: a systematic review. Front. Psychol. **10**, 671 (2019)
29. Mendoza, I., Kalinowski, M., Souza, U., Felderer, M.: Relating verification and validation methods to software product quality characteristics: results of an expert survey. In: Software

Quality: The Complexity and Challenges of Software Engineering and Software Quality in the Cloud: 11th International Conference, SWQD 2019, Vienna, Austria, January 15–18, 2019, Proceedings 11, pp. 33–44. Springer (2019)
30. Molléri, J.S., Petersen, K., Mendes, E.: An empirically evaluated checklist for surveys in software engineering. Inf. Software Technol. **119**, 106240 (2020)
31. Overflow, S.: Stack overflow annual developer survey (2023). https://insights.stackoverflow.com/survey
32. Ralph, P., Baltes, S., Adisaputri, G., Torkar, R., Kovalenko, V., Kalinowski, M., Novielli, N., Yoo, S., Devroey, X., Tan, X., et al.: Pandemic programming: How covid-19 affects software developers and how their organizations can help. Empir. Software Eng. **25**, 4927–4961 (2020)
33. Rea, L.M., Parker, R.A.: Designing and Conducting Survey Research: A Comprehensive Guide. Wiley (2014)
34. Robson, C.: Real World Research: A Resource for social Scientists and Practitioner-Researchers, 2nd edn. Blackwell Publishers, Oxford, UK (2002)
35. Santos, G., Kalinowski, M., Rocha, A.R., Travassos, G.H., Weber, K.C., Antonioni, J.A.: MPS. BR program and MPS model: main results, benefits and beneficiaries of software process improvement in Brazil. In: 2012 Eighth International Conference on the Quality of Information and Communications Technology, pp. 137–142. IEEE (2012)
36. Singer, J., Vinson, N.G.: Ethical issues in empirical studies of software engineering. IEEE Trans. Software Eng. **28**(12), 1171–1180 (2002)
37. Sjøberg, D.I., Dybå, T., Anda, B.C., Hannay, J.E.: Building theories in software engineering. Guide to Advanced Empirical Software Engineering, pp. 312–336 (2008)
38. Stol, K., Ralph, P., Fitzgerald, B.: Grounded theory in software engineering research: a critical review and guidelines. In: L.K. Dillon, W. Visser, L. Williams (eds.) Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14–22, 2016, pp. 120–131. ACM (2016). https://doi.org/10.1145/2884781.2884833
39. Taha, S., Matheson, K., Cronin, T., Anisman, H.: Intolerance of uncertainty, appraisals, coping, and anxiety: The case of the 2009 h 1 n 1 pandemic. Br. J. Health Psychol. **19**(3), 592–605 (2014)
40. Teasdale, E., Yardley, L., Schlotz, W., Michie, S.: The importance of coping appraisal in behavioural responses to pandemic flu. Br. J. Health Psychol. **17**(1), 44–59 (2012)
41. Topp, C.W., Østergaard, S.D., Søndergaard, S., Bech, P.: The who-5 well-being index: a systematic review of the literature. Psychotherapy Psychosomat. **84**(3), 167–176 (2015)
42. Torchiano, M., Fernández, D.M., Travassos, G.H., de Mello, R.M.: Lessons learnt in conducting survey research. In: 5th IEEE/ACM International Workshop on Conducting Empirical Studies in Industry, CESI@ICSE 2017, Buenos Aires, Argentina, May 23, 2017, pp. 33–39. IEEE (2017). https://doi.org/10.1109/CESI.2017.5
43. Torkar, R., Feldt, R., Furia, C.A.: Bayesian data analysis in empirical software engineering: The case of missing data. Contemporary Empirical Methods in Software Engineering, pp. 289–324 (2020)
44. Travassos, G., Kalinowski, M.: iMPS 2013: Evidence on Performance of Organizations That Adopted the MPS-SW Model. Softex, Campinas, Brazil (2014)
45. Wagner, S., Fernández, D.M., Felderer, M., Vetrò, A., Kalinowski, M., Wieringa, R., Pfahl, D., Conte, T., Christiansson, M.T., Greer, D., et al.: Status quo in requirements engineering: A theory and a global family of surveys. ACM Trans. Software Eng. Methodol. (TOSEM) **28**(2), 1–48 (2019)
46. Wagner, S., Mendez, D., Felderer, M., Graziotin, D., Kalinowski, M.: Challenges in survey research. Contemporary Empirical Methods in Software Engineering, pp. 93–125 (2020)
47. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering. Springer Science & Business Media (2012)
48. Yamane, T.: Statistics: An Introductory Analysis. Longman (1973)

Teaching Literature Reviewing for Software Engineering Research

Sebastian Baltes and Paul Ralph

Abstract The goal of this chapter is to support teachers in holistically introducing graduate students to literature reviews, with a particular focus on secondary research. It provides an overview of the overall literature review process and the different types of literature review before diving into guidelines for selecting and conducting different types of literature review. The chapter also provides recommendations for evaluating the quality of existing literature reviews and concludes with a summary of our learning goals and how the chapter supports teachers in addressing them.

1 Introduction

Literature reviews are a crucial part of academic research. Although this is apparent to experienced researchers, it might not be immediately clear to early-stage graduate students. To motivate the topic, a teacher might start with a rhetorical question:

> *Have you ever written a paper for a seminar, a thesis, or a research project? If so, you probably already have done a literature review.*

The ad hoc reviews students typically perform for seminars and theses are usually informal. It is important to motivate that such rather unstructured ad hoc literature reviews might be acceptable for these contexts, but rigorous scientific research requires more systematic approaches.

After introducing the topic of literature reviews, one can take a step back and talk about different ways of doing research. In general, we can stratify scholarship and scholarly writing into *primary*, *secondary*, and *tertiary* research. This distinction

S. Baltes (✉)
University of Bayreuth, Bayreuth, Germany
e-mail: sebastian.baltes@uni-bayreuth.de

P. Ralph
Dalhousie University, Halifax, NS, Canada
e-mail: paulralph@dal.ca

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2024

D. Mendez et al. (eds.), *Handbook on Teaching Empirical Software Engineering*,
https://doi.org/10.1007/978-3-031-71769-7_19

is crucial to understanding the different types of literature review that can be performed.

Primary Research

Primary research involves making observations in the broadest sense of collecting data about objects that are not studies themselves. Computing code metrics, administering questionnaires, interviewing participants, counting bugs, collecting documents, downloading source code, and taking field notes while observing a retrospective meeting are all primary research.

Ideally, different research groups study the same phenomenon independently, allowing other researchers to collect, compare, and aggregate individual results. Since these researchers did not conduct the study themselves but reused their results, this process is called *secondary research*.

Secondary Research

Secondary research involves analyzing, synthesizing, and critiquing primary studies. Ad hoc reviews, case surveys, critical reviews, meta-analysis, meta-synthesis, and scoping reviews are all types of secondary research.

Secondary research is central to evidence-based practice [17]. On the one hand, important decisions should typically be made based on the balance of evidence rather than a single study. On the other hand, from the perspective of a practitioner, reading every study on a topic is often infeasible. Secondary research can help solve this discrepancy. Another issue is that in many fields, individual studies are too small to produce accurate estimates of population parameters; for example, to assess the strength of the relationship between two variables. However, not every paper that somehow combines the results of multiple other papers is automatically of value. Some secondary research degrades into predominately descriptive "papers about papers" with limited scope and usefulness.

Eventually, the body of knowledge on a certain phenomenon or a group of phenomena might be large enough to enable *tertiary research*.

Tertiary Research

Tertiary research has two related meanings: (1) analyses of groups of secondary studies (meta-reviews) and (2) summaries or indices of broad areas of scientific knowledge as found in textbooks, encyclopedia entries, etc.

Although tertiary studies can be valuable in specific circumstances, we should expect them to be rare.

In summary, literature reviews are crucial for understanding, structuring, and synthesizing collective knowledge on a certain topic, making empirical results more accessible to researchers and practitioners. This is particularly true for topics discussed in the software engineering research community.

The purpose of this chapter is to outline how to introduce students to literature reviews, with a particular focus on secondary research. We provide an overview of the different types of literature review before diving into guidelines for selecting and conducting the appropriate literature review type depending on the research context. We also want to support teaching students to evaluate the quality of existing literature reviews and conclude the chapter with a summary of our learning goals and how the chapter supports teachers in addressing them.

2 Types of Literature Review

This section provides a broad overview of the literature review landscape, including references for further reading. As mentioned in the introduction, the first literature review performed by students is usually ad hoc. Although such reviews might be appropriate for seminar papers or theses, it is nonetheless important to understand their limitations. This section is partially based on a previously published short paper that motivates the need for more mature secondary research in software engineering [31]. We are convinced that educating graduate students is an essential step toward achieving that goal.

In-class Suggestion

Starting the lecture with all *types of literature review* might overwhelm students. An alternative to this bottom-up approach would be to start top down; that is, first outline the *process* (see the in-class suggestion in Sect. 3.1), and then provide detailed information on the types. The types themselves can be introduced starting with examples that are aligned with the groups' interests and backgrounds.

In the following, we introduce the seven types of literature review that are common in software engineering research. Tables 1 and 2 summarize the content of this chapter.

Table 1 The seven types of literature review: categorization

Type	Systematic	Purpose	Primary studies	Analysis
Ad hoc review	No	Discuss	Any	Any
Meta-analysis	Yes	Explain and predict	Quantitative	Quantitative
Meta-synthesis	Yes	Explain	Qualitative	Qualitative
Case survey	Yes	Explain and predict	Qualitative	Quantitative
Critical review	Yes	Prescribe	Any	Any
Scoping review	Yes	Describe	Any	Both
Rapid review	Yes	Explain and Predict	Quantitative	Quantitative

Table 2 The seven types of literature review: comparison

Type	Approach
Ad hoc review	Discuss purposively selected related work.
Meta-analysis	Estimate effect sizes by aggregating results of similar quantitative studies.
Meta-synthesis	Synthesize the findings of numerous studies using qualitative analysis.
Case survey	Test causal hypotheses by aggregating case study results.
Critical review	Defend a position and make recommendations by analyzing a sample of papers.
Scoping review	Describe an area of research and map studies into meaningful categories.
Rapid review	A meta-analytic review that compromises rigor for speed.

Ad Hoc Review

An *ad hoc* literature review is simply a discussion of unsystematically selected literature, as contained in most research papers as part of a background or related work section.

Ad hoc reviews may develop theory [30] or integrate a new theory into existing literature [41]. They can further support a position paper or tertiary scholarship. Ad hoc reviews use purposive sampling [3]; that is, researchers purposefully select papers or studies that are useful, relevant, or support their arguments. Ad hoc reviews are often appropriate, for example, to support theory development, identify promising research topics, or prepare for a comprehensive exam. However, they also come with major limitations.

! Attention

Ad hoc reviews suffer from important limitations. Their unsystematic nature introduces sampling bias and defies replication.

An unsystematic, ad hoc approach may lead to cherry-picking of evidence supporting authors' arguments [12]. Therefore, ad hoc reviews are inappropriate for supporting empirical statements such as "x causes y," "most people/objects have property P," or "process P has structure S or follows rules R." Those limitations are best understood by contrasting ad hoc reviews with the various systematic approaches that exist, which we will introduce next.

Before we dive into the different types of *systematic reviews*, we want to clarify the term. Historically, its meaning differs between research communities. The term can be used to describe a systematic review that uses meta-analysis of quantitative studies (especially experiments) to assess the strength of the evidence for specific, usually causal, propositions. In the context of this chapter, to avoid confusion, we refer to this type of review as *meta-analysis* and define *systematic review* as follows.

Systematic Review (Definition)

A *systematic review* is a literature review that employs a systematic (hence the name), replicable process of selecting primary studies for inclusion, including case surveys, critical reviews, meta-analyses, meta-syntheses, and scoping reviews.

This double meaning is due to the history of systematic reviews. The concept of a meta-analytic systematic review emerged from health and medicine in the late twentieth century [28], when practitioners could not keep up with the accelerating production of research. When Chalmers [7] founded the Cochrane Library in 1993, the medical community coalesced around using meta-analysis to inform evidence-based practice. Now, *systematic review* is often conflated with *meta-analysis*. However, other types of systematic reviews have also been around for decades. Scoping reviews were proposed no later than 2005 [1]. Meta-synthesis goes back at least to [15]. Case surveys were proposed as far back as [19].

Systematic Review (Process)

Systematic reviews begin by applying a search strategy to identify primary studies that meet pre-established criteria. Most types of systematic reviews seek to identify *all* the primary studies that meet the selection criteria by combining various techniques to mitigate sampling bias and publication bias as part of the search strategy. Guidelines for developing search strategies are available.

The *ACM SIGSOFT Empirical Standards for Software Engineering*[1] list various techniques to mitigate sampling bias and publication bias [32], which researchers can adopt when developing their search strategy:

- Backward and forward snowballing searches.
- Checking profiles of prolific authors in the area.
- Searching both formal databases (e.g., DBLP) and indexes (e.g., Google Scholar).
- Searching for relevant dissertations.
- Searching pre-print servers (e.g., arXiv).
- Soliciting unpublished manuscripts through mailing lists or social media.
- Contacting known authors in the area.

This systematic search yields a list of primary studies. The way these primary studies are then analyzed determines the *type* of systematic review. If the literature review includes gray literature such as blog posts and whitepapers, it is sometimes referred to as a *multivocal literature review* [10].

2.1 Meta-analysis

Although systematic reviews can be conducted for various types of primary studies, an archetypal systematic review analyzes a set of randomized controlled experiments with the same independent and dependent variables. *Meta-analysis* is then used to aggregate the results of these primary studies.

Meta-analysis

A *meta-analysis* [11] analyzes a set of quantitative studies, usually randomized controlled experiments, with the same independent and dependent variables to statistically aggregate the results of the primary studies into a global effect size estimate.

Consider the following scenario: ten different experiments randomly assigned software engineering undergraduate students to a control group (who complete tasks individually) or a treatment group (who complete tasks in pairs) to compare individual vs. pair programming. The dependent variable was the number of tasks completed successfully. Each primary study reports the results of an independent samples t-test including the mean and standard deviation for each group, the t-statistic, the p-value, Cohen's D (effect size), and the 95% confidence interval for D. Our aim is to combine the results of these ten experiments to estimate the effect

[1] https://www2.sigsoft.org/EmpiricalStandards/docs/standards

of pair programming on performance. Suppose that four studies found a negative effect, three found no significant effect, and three found a positive effect. Can we conclude, based on these study results, that the effect is negative?

Attention

Do not use vote counting to aggregate the results of primary studies, for example, by concluding that an effect is negative because the studies reporting negative results outnumber the studies reporting positive results.

In our scenario, counting votes would mean concluding that the effect is negative because four negative results outweigh three positive results. Vote counting is invalid because primary studies can have wildly different sample sizes and quality levels. What if the studies that found positive effects were much larger and more rigorous while the studies that found negative effects were small and confounded? What if when the three studies without significant results are aggregated, together, their results are significant?

Instead of vote-counting, we apply meta-analysis [11]; that is; we statistically aggregate primary study results into a global effect size estimate. This is often possible with the summary data reported in papers, without the original datasets. Meta-analysis can aggregate results from other kinds of (quantitative) methods as long as the studies have the same independent and dependent variables or overlapping sets of variables. The more complicated the overlaps, the more complicated the meta-analytic model. A comprehensive tutorial on statistical procedures for meta-analysis is beyond the scope of this paper but is readily available [4].

Meta-analytic reviews essentially have the same research question as the studies being reviewed. The purpose of the meta-analysis is to reach a more reliable and robust conclusion by aggregating all available data, implying two important criteria for meta-analysis:

1. Researchers should go to great lengths to find **all** relevant studies.
2. Researchers **must** evaluate the quality of each primary study and either exclude low-quality studies or include study quality as a covariate in the meta-analytic model.

In summary, when scientists equate systematic reviews with evidence-based practice, they usually mean meta-analytic reviews. Meta-analysis aggregates quantitative studies that investigate the same or overlapping hypotheses. *They do not simply describe* existing research. Meta-analysis is rare in software engineering. While there are several good examples [13, 29, 38], quality meta-analysis is dwarfed by superficial scoping reviews [8].

2.2 Meta-synthesis

Since quantitative and qualitative research are both common in the software engineering research community, we want to introduce the analogue of meta-analysis for qualitative research: *meta-synthesis*.

Meta-synthesis

Meta-synthesis refers to a family of methods of aggregating qualitative studies [15]. After identifying the primary studies, the researcher applies hermeneutical and dialectical analyses to understand each primary study, translate them into each other, and construct an account of the body of research; for example, a theory of the central phenomenon that unites the primary studies.

Other names for *meta-synthesis* are thematic synthesis, narrative synthesis, meta-ethnography, and interpretive synthesis. Meta-synthesis requires expertise in qualitative methods and familiarity with the underlying philosophical assumptions. Without a deep understanding of hermeneutical [see 33] and dialectical [see 20] analyses, one should not attempt to perform meta-synthesis. In principle, meta-synthesis can be applied to both qualitative and quantitative work. In practice, such combinations are philosophically strained.

! Attention

Meta-synthesis is **not** organizing papers into categories (as in scoping reviews). Meta-synthesis is the process of synthesizing a credible, nuanced account of a phenomenon from prior qualitative findings.

2.3 Case Survey

A *case survey's* primary studies are (typically qualitative) case studies in the broadest sense (i.e., a scholarly account of some events). Experience reports and gray literature may or may not be included, depending on the study's purposes.

Unlike meta-synthesis, however, a case survey transforms qualitative accounts into a quantitative dataset that supports null-hypothesis testing. Case surveys share the philosophy of meta-analysis (positivism), not meta-synthesis (constructivism) [5].

Case Survey

A *case survey* transforms the results of (typically qualitative) case studies into a quantitative dataset that supports null-hypothesis testing.

Case surveys typically begin with a priori hypotheses and an a priori coding scheme. The researcher reads each case and extracts data into the coding scheme, often using simple dichotomous variables like "Did the team have retrospective meetings? [yes/no]" or "Does the case mention coordination problems? [yes/no]." The resulting dataset is often too sparse for regression modeling, so researchers use simple bivariate correlations to test hypotheses [5].

The Rand Corporation proposed case surveys as a "way to aggregate existing research" [19]. Case surveys were quickly picked up by Yin [46], and later elaborated in management [5, 18]. Today, case survey (a.k.a. case meta-analysis) is widely used in management and information systems research [16]. Although software-engineering-specific case surveys are available [21, 25], they remain rare. However, case surveys have been used to investigate strategic pivots in software start-ups [2], how organizations select component sourcing options [27] and the effectiveness of Distributed Scrum [37]. Case surveys have great potential in software engineering research because case studies are so common.

2.4 Critical Review

The term *critical review* has different meanings in different research communities. We focus on its meaning in software engineering research.

Critical Review

A *critical review* analyzes a sample of qualitative or quantitative primary studies to support an argument or critique, often of a meta-scientific nature.

For example, Stol et al. [41]'s critical review of the use of grounded theory in software engineering *criticizes* method slurring; that is, claiming to have used a research methodology that was not actually used to create illusory legitimacy. Similarly, Baltes and Ralph's critical review [3] *criticizes* how software engineering researchers often overstate sample representativeness and conflate random sampling with representative sampling. In fact, critical reviews in software engineering often investigate methodological topics such as how ethnography is reported [47] or how qualitative research is synthesized [14]. Critical reviews differ from case surveys and meta-analyses in two important ways.

First, meta-analytic reviews aggregate evidence on causal relationships to generate evidence-based recommendations, while critical reviews critically evaluate (methodological) issues. Critical reviews are not done to support evidence-based practice or to summarize evidence for a theory. Critical reviews are part of the *meta-scientific discourse*; that is, a scientific community's internal conversations about how it conducts research.

Second, for many critical reviews, including *all* relevant primary studies is impossible and unnecessary. For example, a critical review of adherence to the *Introduction, Method, Results and Discussion* (IMRaD) framework [39] could include all software engineering papers ever written. Instead, a random sample of papers from a selection of leading journals and conferences is sufficient because critical reviews do not assess causal claims; therefore, publication bias—"What if significant results were published but non-significant results were not?"—is irrelevant.

Analysis performed within a critical review can be quantitative, qualitative, or both. However, critical reviews typically adopt a *critical* stance; that is, they go beyond mere description and offer specific critiques of the work being reviewed.

2.5 Scoping Review

What is often called a *systematic mapping study* in software engineering [26] is generally called a *scoping review* elsewhere; for example, in health, medicine, and psychology.

Scoping Review

The purpose of a *scoping review* is to understand the state of research on a particular topic, typically by mapping primary studies into categories.

Scoping reviews are primarily descriptive; they count the number of studies on a topic. They often organize studies by research method, subtopic, authors, geographical location, publication venue, etc. They often conclude that more research is needed on particular subtopics. For example, Mohanani et al. [22] mapped primary studies according to which cognitive bias (e.g., confirmation bias) they investigated and in which area of software development (e.g., design, management) they investigated it, and then called for more research on *debiasing*; that is, preventing or mitigating cognitive biases.

! Attention

The problem with scoping reviews is that they typically do not meet any of the core purposes of secondary research.

When comparing *scoping reviews* to *meta-analyses* and *case surveys*, one notices that the latter synthesize the results of many studies to answer specific empirical (often causal) questions about the world. Scoping reviews include a similar search, but typically do not provide sufficient quantitative synthesis to answer important empirical questions. Therefore, scoping reviews do not inform evidence-based practice as meta-analyses and case surveys do. *Meta-synthesis* involves deep, theory-oriented reinterpretation of related qualitative studies. Although scoping reviews can include qualitative analysis (e.g., mapping or categorization), that analysis is often too superficial to generate novel and useful theories.

Critical reviews use a sample of papers to demonstrate an important pattern for the internal discourse of a scientific community. While scoping reviews often give recommendations regarding future research, they focus on an empirical topic (e.g., cognitive biases in SE), not a meta-scientific topic (e.g., construct validity); therefore, they are not configured, from the outset, to deliver useful meta-scientific critique.

In summary, scoping reviews begin like other kinds of systematic reviews but stop short of *synthesizing* the data into aggregate empirical results, theory, or meta-scientific critique. This is by definition: If a scoping review applies a meta-analytic model to aggregate primary study results or applies hermeneutics and dialectics to synthesize qualitative accounts or develops an evidence-based critique of a scientific practice, it is no longer a scoping review; it is a meta-analysis, a case survey, a meta-synthesis, or a critical review. Therefore, some authors recommend a scoping review "as a precursor to a systematic review" [23]. Ideally, then, scoping reviews would be published as works-in-progress (e.g. as posters or workshop papers) rather than as full-length journal articles. We encourage teachers to build awareness that, while many scoping reviews have been published in leading software engineering venues, this is perhaps not commensurate with their epistemic value or the norms of other scientific communities.

2.6 Rapid Review

Systematic literature reviews can be very time consuming; for example, 18 months to complete and another year or more to publish. In some situations, decision-making cannot wait until this time-consuming process is finished.

Rapid Review

A *rapid review* is a meta-analysis that makes methodological compromises to reduce completion time [9].

Ganann et al. found many such compromises including restricting the literature search, truncating results, omitting techniques for overcoming publication bias (e.g., reference snowballing), streamlining screening and data extraction, and skipping quality assessment [9]. Some authors argue that rapid reviews are a suitable means for practitioners to provide evidence in a timely manner [6]. However, from a scientific perspective rapid reviews are justified *if and only if* evidence is needed to support imminent decisions, and waiting for a comprehensive meta-analytic review would be harmful. These conditions occur in health and medicine, for example, when an unprecedented viral pandemic strikes. These conditions rarely occur in software engineering.

Attention

The term *rapid review* should not be used to legitimize bad systematic reviews where there is no urgent need for immediate results.

3 Guidelines

After the previous section introduced different types of literature review, this section provides advice for selecting, performing, and evaluating them. We start by describing the overall literature review process (Sect. 3.1) and then continue to discuss how students can use *ad hoc reviews* and *scoping reviews* to screen literature (Sect. 3.2) and how they can subsequently reflect on which secondary research method might be appropriate (Sect. 3.3). We continue with recommendations on performing secondary research (Sect. 3.4) and conclude with advice on evaluating literature reviews (Sect. 3.5). Rather than comprehensive guidelines, we outline the main pitfalls as well as anti-patterns, and highlight crucial aspects.

Learning Goals

The learning goals of a lecture on literature reviews are that students: (1) understand the overall process, (2) can conduct ad hoc and scoping reviews independently, (3) are aware of existing secondary research methods, and (4) can evaluate existing literature reviews. Actually performing secondary research is out of scope for such a lecture. If a teacher wants to expand the scope of the lecture, we suggest focusing on *case surveys*, which are simpler than *meta-analysis* or *meta-synthesis*.

3.1 The Literature Review Process

Figure 1 outlines the process of selecting appropriate literature review types at different stages of a student's research project. Students often begin with an ad hoc review to elaborate and assess a rough idea, informally gathering a first overview of existing related work. This information can then be used to formulate more specific research questions, which could also be defined without first conducting an ad hoc review. Having defined specific research questions, the next step would be to perform a *scoping review* to thoroughly understand the state of research on the topic of interest.

In-class Suggestion

In class, the teacher can demonstrate how tools such as Google Scholar and DBLP can be used for *ad hoc reviews*. The teacher can then share a research idea that students are asked to explore in a *buzz group* setting. The resulting papers are jointly collected, e.g., via Etherpad. Based on the resulting list, the teacher can discuss additional aspects of ad hoc reviews, such as selecting only peer-reviewed articles.

After the initial screening phase is completed, a researcher might decide that no further literature review is required. The next steps would then be to plan and conduct a novel study against the background of the identified related work. Reasons for not conducting a *scoping review* after the *ad hoc review* include:

- The *ad hoc review* yielded existing studies that are so close to the research idea or research questions that one can either continue planning a replication study or decide that the idea is not worth pursuing.
- The *ad hoc review* yielded recent *scoping reviews* on the topic that can be used to further plan the intended study.
- The *ad hoc review* yielded recent *secondary research* on the topic that needs further screening.
- The research question is of meta-scientific nature and warrants a *critical review* (see Sects. 2.4 and 3.4.4).

It is important to explain to students that a large research project often involves a *scoping review*. Finding very similar related work late in the research process, limiting the novelty of an ongoing study, is very demotivating, not only for students.

Critical reviews are a special case, because they are often performed on a random sample of papers selected from specific venues in a defined period of time. As motivated in Sect. 2.4, including all relevant papers is often infeasible for critical reviews, and therefore a scoping review is not a logical predecessor of a critical review.

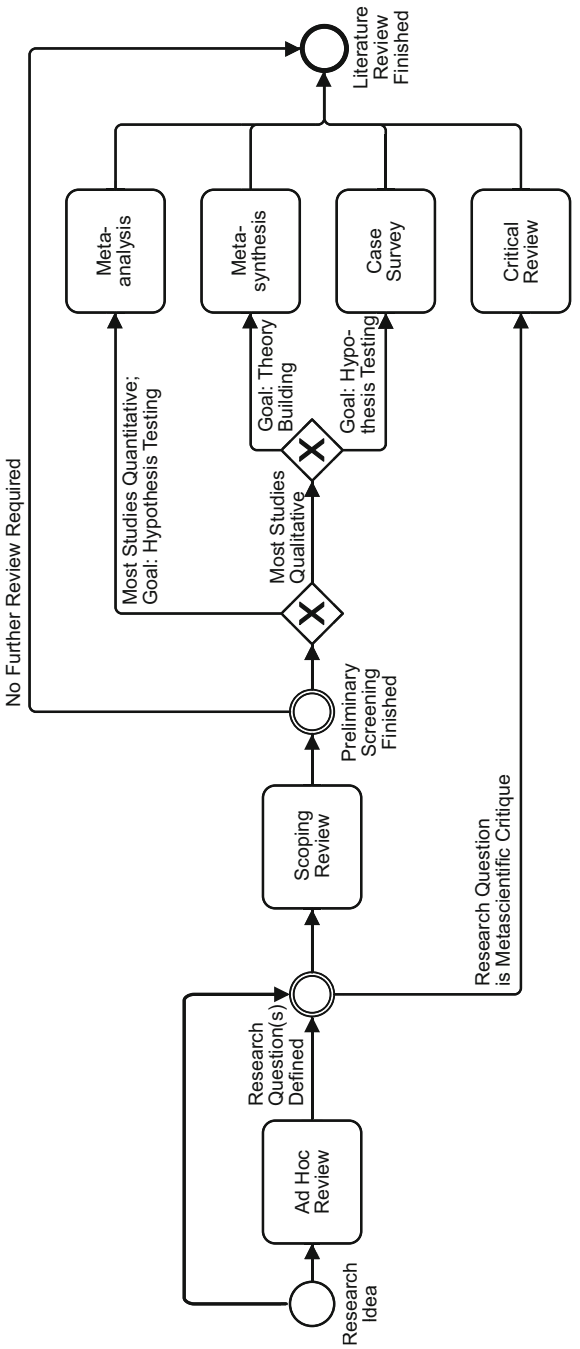


Fig. 1 Process of selecting literature review types at different stages of a research project

If the *ad hoc review* warrants performing a *scoping review*, the researcher must screen the collected articles and decide which type of secondary research is suitable (Sect. 3.3), or if the scoping review itself is sufficient. Reasons for not conducting *secondary research* include:

- The research methods of the identified papers are too diverse.
- The research questions of the identified papers are not aligned.
- Papers lack statistics or other information required to perform secondary research.
- Not enough relevant papers are available.

In-class Suggestion

To underline the *diversity of research methods and ways of presenting empirical studies in software engineering*, the teacher might contrast papers on human aspects of software engineering, mining software repositories, and software testing (to name a few examples). Papers in psychology or medical research can serve as examples for disciplines with more uniform research methods and a more standardized structure for presenting empirical studies.

3.2 Screening Literature

In this section, we focus on the screening phase of the process outlined in Fig. 1. What both *ad hoc reviews* and *scoping reviews* have in common is that they should start with a clearly formulated research idea, purpose, goal, or question (see also the chapter *Research Design in Software Engineering* by Molléri and Petersen).

In-class Suggestion

Consider providing a counterexample; that is, a vague and ambiguous *research goal*, and then refine it together with the students.

Based on the research idea or question, the next step is to define a search strategy that typically includes an initial set of keywords, inclusion/exclusion criteria, and the search engines to use. Popular choices for computer science research are Google Scholar and DBLP. Semantic Scholar is another freely available search engine. Commercial options include Web of Science and Scopus.

Some portals support *backward and/or forward snowballing*. Backward snowballing refers to analyzing the papers that an included paper cites, while forward snowballing refers to analyzing papers that cite an included paper. Guidelines for

snowballing in literature reviews [43] and for combining keyword-based portal searches with snowballing [44] are available.

The inclusion and exclusion criteria define which paper the researchers are interested in. Typical filters restrict the time span (e.g., only papers published in the last 5 years) or exclude articles not meeting certain standards (e.g., non-peer-reviewed articles or book chapters). If keywords have different meaning in different contexts/disciplines, the inclusion/exclusion criteria can clarify the instances that the researchers want to consider.

In-class Suggestion

Based on the previously defined research goal, the teacher can guide students through defining an initial set of *keywords and inclusion/exclusion criteria*. It is important to stress that both the keywords and the criteria can evolve over the course of a research project. However, all changes must be documented.

Conducting ad hoc or *scoping reviews* involves reading many paper titles, abstracts, and sections. It should be stressed how important deep reading is for research. Knowing when to skim over (parts of) a paper is a skill that students develop over time. In the beginning, students should err on the side of reading more and skimming less, even if it slows them down. We are well aware of the pressure to publish early and often, but this is hard to do without a deep understanding of the field. Part of that understanding is, besides published research articles, being able to distinguish scientific evidence from opinionated articles, "laws," and recommendations. Interesting examples in this regard are the *Gartner Hype Cycle*, which claimed that software engineering was at its "peak of inflated expectations" in 2023, over 55 years after the first NATO Conference on Software Engineering [24], or the consulting company McKinsey's claim that their consultants can measure software developer productivity, despite scientific research suggesting a much more nuanced view [36]. Finally, students must understand that ad hoc and *scoping reviews* are usually not a study on their own but a means of preparing themselves to perform novel studies or replications.

Exercise Suggestion

An exercise could be to let students conduct a limited *ad hoc review* at home, using the portals, keywords, and criteria discussed during the lecture. Identifying a fixed number of papers (e.g., up to 20) and then summarizing a subset (e.g., three of them) in their own words, without repeating sentences from the abstracts, could be reasonable guardrails. Bonus exercise: include snowballing.

3.3 Reflecting on Screened Literature

In this section, we focus on the decision points in Fig. 1. In Sect. 3.1, we already described in which cases one might not perform a *scoping review* after an *ad hoc review* or not perform *secondary* research after a *scoping review*. Here, we will focus mainly on the selection of appropriate secondary research methods.

As mentioned above, *critical reviews* are a special case. They are meta-scientific in nature and analyze a sample of qualitative or quantitative primary studies to support an argument or critique (see Sects. 2.4). They would usually not be done after a scoping review, but instead of one. We do not expect students to conduct critical reviews early in their careers, and therefore we only refer to our recommendations in Sect. 3.4.4.

Assuming that the students have conducted a *scoping review*, the question arise: how do they decide whether the corpus of identified articles warrants secondary research. If there is indeed a considerable body of knowledge, most studies are quantitative, and the goal is to test one or more hypotheses, the next step would be to perform *meta-analysis* (see Sects. 2.1 and 3.4.1). If most studies are qualitative and the goal is theory building, performing *meta-synthesis* is possible (see Sects. 2.2), given that the researcher has the appropriate expertise and experience (see Sect. 3.4.2). If most articles report case studies and the objective is to test hypotheses, students might consider conducting a *case survey* (see Sects. 2.3 and 3.4.3).

In a lecture on literature reviews, the concepts of *meta-analysis* and *meta-synthesis* can be presented, but we consider *case surveys* the most approachable type of secondary research for software engineering graduate students. In the following section, we elaborate on this further.

In-class Suggestion

Teachers can stress that, unfortunately, *good secondary research is rare in software engineering*. In class, best-practice examples from software engineering can be shown along with examples from other fields.

3.4 Literature Review Recommendations

As mentioned above, performing *meta-analysis* or *meta-synthesis* will probably be too challenging for most graduate students. Performing *critical reviews* as a student is also rare. We provide recommendations for all of these types of secondary research but would like to stress that besides conducting *ad hoc reviews* and *scoping reviews*, *case surveys* are most appropriate for software engineering graduate students, and hence teachers should focus on those three methods while outlining the others.

In-class Suggestion

Focus on *ad hoc reviews*, *scoping reviews*, and *case surveys* but also summarize *meta-analysis* and *meta-synthesis*, making clear that these are advanced secondary research methods usually out of scope for graduate students in software engineering.

3.4.1 Meta-analysis

In our experience, the overwhelming problem with *meta-analysis* in software engineering is that, for a given topic, we almost never have enough similar quantitative studies to aggregate statistically. The research questions and statistical methods are usually too diverse, or essential properties such as effect sizes are missing. Moreover, software engineering research can be performed in very diverse contexts, further limiting the comparability of studies. Therefore, an attempt to conduct a true meta-analysis almost always devolves into a scoping review.

In addition, meta-analysis requires multiple researchers to perform the abstract-and-title screening and then the full-text screening. They need to double-enter all the data, which subsequently needs to be compared and aligned. Hence, this is not something a graduate student can and should do alone. One needs a team.

Moreover, meta-analysis is very time-consuming and often takes more than a year. This results in pragmatic issues such as new papers being published in the process of performing the meta-analysis, but also during the paper review process after submission. The dataset must be constantly kept up-to-date, and the statistical methods used in the meta-analysis should be automated so that statistics can be regularly updated.

Our specific recommendations for students (and supervisors) who nevertheless want to conduct a meta-analysis are:

- Read textbooks dedicated to the topic (e.g., [4]). A solid background in statistics is helpful.
- Do not vote-count (see Sect. 2.1).
- Statistically model study quality instead of excluding low-quality studies. Refer to the empirical standards [32] to assess study quality.

In summary, the situation around meta-analysis in software engineering is a chicken-and-egg problem. We are convinced that software engineering needs more meta-analysis, but this kind of secondary research is very hard to do, and the success of students in that endeavor is improbable, because research and reporting quality in software engineering do not meet the standards of other disciplines such as medical research or psychology.

3.4.2 Meta-synthesis

Meta-synthesis requires a strong appreciation of qualitative research. Crucial to this is understanding constructivism and interpretivism. Without this understanding, meta-synthesis is virtually impossible. Moreover, meta-synthesis should only be attempted by people with considerable experience in conducting qualitative studies. Again, without such experience, meta-synthesis is virtually impossible.

Ideally, to do meta-synthesis well, researchers need to understand the differences between different qualitative research traditions (e.g., case study, ethnography, grounded theory, phenomenology, critical theory). Very few researchers in software engineering have such a deep methodological understanding because the methods are rooted in other disciplines; hence, "method slurring" is common [41]. Although evidence standards [32] can be used to assess study quality, assessing the quality of qualitative research can be more difficult because there are not as many obvious red flags, such as missing effect sizes. All this further complicates meta-synthesis.

Finally, when performing a meta-synthesis, one must organize the findings of the primary studies into reasonable narratives without forcing one's own expectations or assumptions onto the data. Auto-reflection [42] is crucial. In summary, our recommendation is to only attempt meta-synthesis after:

- Reading *many* books on qualitative methods.
- Conducting several qualitative studies.
- Reading at least one book on meta-synthesis (e.g., [15]).
- Reading *many* studies from a range of qualitative research traditions.

Unfortunately, having a teacher who has done all of the above is not enough. The primary analyst needs the expertise that comes from a combination of firsthand experience and secondhand study. Therefore, meta-synthesis is not a great choice for most graduate students unless an experienced supervisor is *very* engaged in the analysis.

3.4.3 Case Survey

In our opinion, *case surveys* are a great option for graduate students because case surveys are more structured than *meta-synthesis* but less difficult to perform than *meta-analysis*. One does not need to know as much about research methods as for a *critical review*. In our experience, case study is one of the most common research methods in software engineering, so there are a lot of good—and not so good—examples out there.

One problem is that, as mentioned in the previous section, method slurring is a problem in software engineering research, and this also applies to *case studies*. As a researcher, one needs to have clear rules to decide what a case study is (to include it) and what is not (to exclude it). One cannot make this decision based on the presence (or absence) of the words "case study" alone. There are different schools of thought on what constitutes a *case study*.

Yin [45] argued that a case must have clear evidence of triangulation across different kinds of data (e.g., interviews, online discussion threads, change logs), whereas Bullock and Tubbs [5] included just about any account of the phenomenon of interest. Another way is to include only articles that clearly identify a *site* that the researchers visited. This third way is good for excluding a lot of low-quality pseudo-cases but is problematic for studies of remote-first teams and historical cases.

A problem is that while guidelines exist [34, 35], software engineering research does not have the history of rich description in case study research that, for example, sociology has. Therefore, students might run into the problem that the analyzed articles do not report the required details.

In-class Suggestion

Given the large number of case studies in software engineering research and the existence of guidelines and examples, teachers should focus on this secondary research method and present the existing guidelines and best-practice examples.

For *case surveys*, the statistics are simpler than for *meta-analysis*, because the resulting dataset is much more sparse and statistics is limited to fundamental methods such as correlations. Structural equation modeling (SEM) or partial least squares regression (PLS) is usually not possible to use for the reasons outlined above.

Our recommendation is that all data extraction should be performed by two researchers independently and then compared and aligned to ensure quality and assess reliability. Bullock and Tubbs [5] recommend asking all the primary study authors to fill out the data collection for their papers in a survey, so that they act as a second coder and can enter information they know but was not reported in the article. However, it is not easy to motivate the authors to do this (or to contact them in the first place), especially for older papers. It might help if not the student but a senior supervisor that is well-known in the field sends out the invites, but even then the response rate will most likely be far from 100%.

3.4.4 Critical Review

In theory, *critical reviews* can be a great choice for a graduate student because doing a critical review usually means analyzing the research methods used in an area, finding all the common problems, and figuring out how to do a method better. This can be a great prelude to conducting one's own research because it helps the student avoid all these pitfalls they identified in the *critical review* and hence design better studies.

However, *critical reviews* suffer from some of the same problems as *meta-analysis*—you need a team, and the analysis can be time-consuming. Moreover, performing a *critical review* on a method that the student has not or only rarely

used themselves is challenging. An advantage of *critical reviews* is that they do not require a comprehensive sample that includes—in the best case—all studies on a topic. One only needs a defensible sample of studies. Therefore, the sampling and analysis of papers does not take as long as for *meta-analysis* or *meta-synthesis*, and a slightly outdated sample does not affect the *critical review* as much.

The biggest challenges we have encountered with critical reviews are that:

- Most students (and also many supervisors) have insufficient research methods training to assess the primary studies. One reason for that is the diversity of research methods in software engineering, as explained above.
- Reviewers may be unfamiliar with *critical reviews* and therefore treat them as scoping reviews, expecting the authors to include all relevant studies. As described in Sect. 2.4, this is not required for critical reviews.
- *Critical reviews* have ethical implications (see below).

! Attention

Ethics: It is hard to critique primary studies without publicly blaming the authors. We recommend describing problems without citing the primary studies that exemplify those problems. One can state explicitly that this is done because the additional credibility of pointing to examples does not justify the hurt feelings of the authors and the potential impact on the reputation of the researchers conducting the review. A risk for us as authors is that reviewers might not accept this argument.

3.5 Evaluating Literature Reviews

Most of the time, students will read or review literature reviews at some point during their studies. Therefore, we want to teach them which aspects to look out for when assessing the quality of literature reviews.

Learning to assess the quality of existing research is a crucial skill for any scientist. We cannot infer that a study is good just because it was peer-reviewed and published in a reputable venue (or bad because it is a preprint or published in an outlet of ill repute). The criteria described in this section are adapted from the *ACM SIGSOFT Empirical Standards for Software Engineering* [32]. Rather than repeating all the criteria, we will focus on the most common red flags.

3.5.1 Evaluating Ad Hoc and Scoping Reviews

An *ad hoc review* is what students would report in the related work section of a research paper. They are not a stand-alone publication. Similarly, *scoping reviews* are basically pilot studies on the way to conducting a real systematic review.

However, given the diversity of research methods and the issues with reporting empirical studies in software engineering, which we described above, they are often published as stand-alone articles in journals or full-paper conference tracks. This is uncommon in other disciplines and will hopefully change as secondary research in software engineering matures [31].

The most common red flag in an *ad hoc review* is the vague feeling that the authors have not actually read, in detail, the papers being discussed. When the reader is familiar with some of the reviewed works and the authors' comments on them suggest serious misunderstandings, the reader worries that the review is superficial.

In contrast, the authors of *scoping reviews* do not necessarily read the primary studies in depth; they rigorously extract specific data from the studies. Some things that suggest low quality in a scoping review are:

- The process of selecting the primary studies is not described in sufficient detail that other researchers could replicate it.
- There is no public dataset.
- The authors did not apply techniques for mitigating sampling bias [3] (e.g., reference snowballing).

3.5.2 Evaluating Meta-analysis

Meta-analysis usually involves estimating the effect size of one or more causal relationships by creating a statistical meta-model to aggregate the results of numerous similar studies. Two of the biggest validity threats to *meta-analysis* are publication bias and the quality of the primary studies. This leads to several common red flags:

- The study does not have a statistical meta-model, or the model does not take into account differences in primary study size and quality. Sometimes, there might be no meta-model because the paper is a scoping review masquerading as meta-analysis.
- Insufficient attempts to find all relevant studies (e.g., overly restrictive search terms, no reference snowballing).
- Insufficient attempts to mitigate and asses publication bias (e.g., including pre-prints and dissertations).
- No discussion of inter-rater reliability for abstract/title and full-text screening.
- Missing or incomplete replication package, including dataset.

For a really quick and dirty assessment, if the meta-analysis paper does not have a PRISMA diagram[2] (showing study selection) or a funnel plot [40] (visualizing publication bias), it is probably of low quality.

[2] https://www.prisma-statement.org/prisma-2020-flow-diagram

3.5.3 Evaluating Meta-synthesis

The synthesis of qualitative research does not involve complicated statistical models, but it does involve a complicated process of comparing, contrasting, and translating concepts and findings between studies. Qualitative studies often use (and invent) different concepts to describe and explain related phenomena, so synthesizing the findings involves retelling one paper's narratives in the languages of other papers. This is much more difficult than it sounds, so a good meta-synthesis explains—*in vivid detail*—how the primary studies were coded, compared, contrasted, translated, and (eventually) integrated into themes. Red flags include:

- The themes are not clearly defined, vividly explained, and grounded in *many* quotations from the primary studies.
- The data analysis process is only briefly or superficially explained.

3.5.4 Evaluating Case Surveys

Case surveys are quite similar to *meta-analysis* except that a case survey's statistical meta-model is much simpler. Each case study has the same size ($n = 1$), so we do not model study size. Since we are only concerned with the facts of the case (not the interpretations), we do not exclude low-quality cases or model study quality unless we have reason to believe a case was fabricated. However, there are red flags to watch out for, including the following:

- Insufficient attempts to find all relevant studies (e.g., overly restrictive search terms, no reference snowballing).
- No discussion of inter-rater reliability for abstract and full-text screening; no discussion of quality control during data extraction.
- Insufficient a priori justification of hypotheses.
- Missing or incomplete replication package, including dataset.

About that last red flag: case surveys lend themselves to *HARKing* (Hypothesizing After Results are Known). In an ideal world, case surveys, including all of their hypotheses, would be publicly pre-registered such that HARKing is easily identifiable and thus discouraged. Without pre-registration, however, we can only look for how well each hypothesis is justified conceptually or based on previous research.

3.5.5 Evaluating Critical Reviews

At risk of repeating ourselves, *critical reviews* do not need to include every study that meets the selection criteria. Critical reviews often just take a (stratified) random sample of papers from some venues the authors consider reputable, and—for the scope of a *critical review*—that is fine, because the point of a critical review is

to make some argument about the way research is done in some area, not to estimate the effect size of a causal relationship. Publication bias is usually irrelevant; therefore, critical reviews do not need PRISMA diagrams or funnel plots. Critical reviews do not exclude low-quality studies because the whole point is to critique the quality of the primary studies.

The quality of a critical review is all about the critique itself. A good critical review identifies several problems with existing research and suggests specific ways to address these problems. Common red flags include the following:

- The critique seems superficial.
- The suggested solutions are intractable.
- The authors of the critical review do not seem to be experts in the method they are critiquing.

An example of the last point: a critical review of experimental designs mixing up quasi-experiments with true experiments would be a serious red flag.

4 Conclusion

As described in Sect. 3, the learning goals of a graduate lecture on literature reviews should be that students:

1. Understand the overall literature review process and can reproduce Fig. 1 capturing that process.
2. Conduct ad hoc and *scoping reviews* according to our suggestions in Sects. 2 and 3.
3. Explain the secondary research methods; that is, they can reproduce the description of *meta-analysis*, *meta-synthesis*, and *case survey*.
4. Evaluate existing literature reviews according to the recommendations in Sect. 3.5.

As mentioned earlier, if a teacher wants to expand the scope of the lecture, we suggest focusing on *case surveys* rather than *meta-analysis* and *meta-synthesis* because case studies are common in software engineering research and the barriers to performing that kind of secondary research are lower for students. We hope that this chapter, together with the in-class and exercise suggestions, will support teachers in preparing a lecture on literature reviews.

References

1. Arksey, H., O'Malley, L.: Scoping studies: towards a methodological framework. Int. J. Soc. Res. Methodol. **8**(1), 19–32 (2005)
2. Bajwa, S.S., Wang, X., Duc, A.N., Abrahamsson, P.: "Failures" to be celebrated: an analysis of major pivots of software startups. Empir. Softw. Eng. **22**(5), 2373–2408 (2017)

3. Baltes, S., Ralph, P.: Sampling in software engineering research: A critical review and guidelines. Empir. Softw. Eng. **27**(94) (2022)
4. Borenstein, M., Hedges, L.V., Higgins, J.P., Rothstein, H.R.: Introduction to Meta-Analysis. Wiley, Chichester (2021)
5. Bullock, R., Tubbs, M.E.: The case meta-analysis method for od. In: Research in Organizational Change and Development, vol 1, pp 171–228. JAI Press, Stamford (1987)
6. Cartaxo, B., Pinto, G., Soares, S.: Rapid reviews in software engineering. In: Felderer, M., Travassos, G.H. (eds) Contemporary Empirical Methods in Software Engineering, pp 357–384. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-32489-6_3
7. Chalmers, I.: The Cochrane collaboration: preparing, maintaining, and disseminating systematic reviews of the effects of health care. Ann. N. Y. Acad. Sci. **703**(1), 156–165 (1993)
8. Cruzes, D.S., Dybå, T.: Research synthesis in software engineering: a tertiary study. Inf. Softw. Technol. **53**(5), 440–455 (2011)
9. Ganann, R., Ciliska, D., Thomas, H.: Expediting systematic reviews: methods and implications of rapid reviews. Implement. Sci. **5**(1), 1–10 (2010)
10. Garousi, V., Felderer, M., Mäntylä, M.V.: Guidelines for including grey literature and conducting multivocal literature reviews in software engineering. Inf. Softw. Technol. **106**, 101–121 (2019). https://doi.org/10.1016/j.infsof.2018.09.006
11. Glass, G.V.: Primary, secondary, and meta-analysis of research. Edu. Res. **5**(10), 3–8 (1976)
12. Gough, D., Oliver, S., Thomas, J.: An Introduction to Systematic Reviews. Sage, London (2017)
13. Hannay, J.E., Dybå, T., Arisholm, E., Sjøberg, D.I.: The effectiveness of pair programming: a meta-analysis. Inf. Softw. Technol. **51**(7), 1110–1122 (2009)
14. Huang, X., Zhang, H., Zhou, X., Babar, M.A., Yang, S.: Synthesizing qualitative research in software engineering: A critical review. In: Proceedings of the 40th International Conference on Software Engineering, Association for Computing Machinery, New York, NY, ICSE '18, pp. 1207–1218 (2018). https://doi.org/10.1145/3180155.3180235
15. Jensen, L.A., Allen, M.N.: Meta-synthesis of qualitative findings. Qualit. Health Res. **6**(4), 553–560 (1996)
16. Jurisch, M.C., Wolf, P., Krcmar, H.: Using the case survey method for synthesizing case study evidence in information systems research. In: Proceedings of the 19th Americas Conference on Information Systems (AMCIS), Association for Information Systems, Chicago, Article 5 (2013)
17. Kitchenham, B.A., Dyba, T., Jorgensen, M.: Evidence-based software engineering. In: Proceedings of the 26th International Conference on Software Engineering, IEEE Computer Society, Washington, D.C., ICSE '04, pp. 273–281 (2004)
18. Larsson, R.: Case survey methodology: quantitative analysis of patterns across case studies. Acad. Manag. J. **36**(6), 1515–1546 (1993)
19. Lucas, W.A.: The case survey method: Aggregating case experience. Technical Report 1515, Rand (1974). https://www.rand.org/pubs/reports/R1515.html
20. Maybee, J.E.: Hegel's dialectics. In: Zalta, E.N. (ed.) The Stanford Encyclopedia of Philosophy, Winter 2020 edn. Metaphysics Research Lab, Stanford University, Stanford (2020)
21. Melegati, J., Wang, X.: Case survey studies in software engineering research. In: Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), Association for Computing Machinery, New York, NY, ESEM '20 (2020). https://doi.org/10.1145/3382494.3410683
22. Mohanani, R., Salman, I., Turhan, B., Rodríguez, P., Ralph, P.: Cognitive biases in software engineering: a systematic mapping study. IEEE Trans. Softw. Eng. **46**(12), 1318–1339 (2018)
23. Munn, Z., Peters, M.D., Stern, C., Tufanaru, C., McArthur, A., Aromataris, E.: Systematic review or scoping review? Guidance for authors when choosing between a systematic or scoping review approach. BMC Med. Res. Methodol. **18**(1), 1–7 (2018)
24. Naur, P., Randell, B.: Software Engineering: Report of a Conference Sponsored by the NATO Science Committee, Garmisch, 7–11 Oct. 1968, Brussels, Scientific Affairs Division, NATO (1969)

25. Petersen, K.: Guidelines for case survey research in software engineering. In: Felderer, M., Travassos, G.H. (eds.) Contemporary Empirical Methods in Software Engineering, pp 63–92. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-32489-6_3
26. Petersen, K., Feldt, R., Mujtaba, S., Mattsson, M.: Systematic mapping studies in software engineering. In: Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering, BCS Learning & Development, Swindon, GBR, EASE'08, pp. 68–77 (2008)
27. Petersen, K., Badampudi, D., Shah, S.M.A., Wnuk, K., Gorschek, T., Papatheocharous, E., Axelsson, J., Sentilles, S., Crnkovic, I., Cicchetti, A.: Choosing component origins for software intensive systems: In-house, COTS, OSS or outsourcing?—a case survey. IEEE Trans. Softw. Eng. **44**(3), 237–261 (2017)
28. Purssell, E., McCrae, N.: How to Perform a Systematic Literature Review: A Guide for Healthcare Researchers, Practitioners and Students. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-49672-2_2
29. Rafique, Y., Mišić, V.B.: The effects of test-driven development on external quality and productivity: a meta-analysis. IEEE Trans. Softw. Eng. **39**(6), 835–856 (2012)
30. Ralph, P.: Toward methodological guidelines for process theories and taxonomies in software engineering. IEEE Trans. Softw. Eng. **45**(7), 712–735 (2018)
31. Ralph, P., Baltes, S.: Paving the way for mature secondary research: the seven types of literature review. In: Roychoudhury, A., Cadar, C., Kim, M. (eds.) Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2022, Singapore, November 14–18, 2022, ACM, pp. 1632–1636 (2022). https://doi.org/10.1145/3540250.3560877
32. Ralph, P., Baltes, S., Bianculli, D., Dittrich, Y., Felderer, M., Feldt, R., Filieri, A., Furia, C.A., Graziotin, D., He, P., et al.: ACM SIGSOFT empirical standards. arXiv: 201003525 [csSE] (2020)
33. Ricoeur, P.: Hermeneutics and the Human Sciences: Essays on Language, Action and Interpretation. Cambridge University Press, New York (1981)
34. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. Empir. Softw. Eng. **14**(2), 131–164 (2009). https://doi.org/10.1007/s10664-008-9102-8
35. Runeson, P., Höst, M., Rainer, A., Regnell, B.: Case Study Research in Software Engineering - Guidelines and Examples. Wiley, Hoboken (2012). http://eu.wiley.com/WileyCDA/WileyTitle/productCd-1118104358.html
36. Sadowski, C., Zimmermann, T. (eds.): Rethinking Productivity in Software Engineering. Apress Open/Springer, Berlin (2019). https://doi.org/10.1007/978-1-4842-4221-6
37. Santos, R.D.S., Ralph, P., Arshad, A., Stol, K.J.: Distributed scrum: a case meta-analysis. ACM Computing Surveys **56**(4), 1–37 (2023). https://doi.org/10.1145/3626519
38. Shepperd, M., Bowes, D., Hall, T.: Researcher bias: the use of machine learning in software defect prediction. IEEE Trans. Softw. Eng. **40**(6), 603–616 (2014)
39. Sollaci, L.B., Pereira, M.G.: The introduction, methods, results, and discussion (IMRAD) structure: a fifty-year survey. J. Med. Library Assoc. **92**(3), 364 (2004)
40. Sterne, J.A.C., Harbord, R.M.: Funnel plots in meta-analysis. Stata J. **4**(2), 127–141 (2004). https://doi.org/10.1177/1536867X0400400204
41. Stol, K.J., Ralph, P., Fitzgerald, B.: Grounded theory in software engineering research: A critical review and guidelines. In: Proceedings of the 38th International Conference on Software Engineering, Association for Computing Machinery, New York, NY, ICSE '16, pp. 120–131 (2016). https://doi.org/10.1145/2884781.2884833
42. Walsh, R.: The methods of reflexivity. Humanist. Psychol. **31**(4), 51–66 (2003). https://doi.org/10.1080/08873267.2003.9986934
43. Wohlin, C.: Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: Shepperd, M.J., Hall, T., Myrtveit, I. (eds.) 18th International Conference on Evaluation and Assessment in Software Engineering, EASE '14, London,

England, May 13–14, 2014, ACM, pp. 38:1–38:10 (2014). https://doi.org/10.1145/2601248.2601268
44. Wohlin, C., Kalinowski, M., Felizardo, K.R., Mendes, E.: Successful combination of database search and snowballing for identification of primary studies in systematic literature studies. Inf. Softw. Technol. **147**, 106908 (2022). https://doi.org/10.1016/j.infsof.2022.106908
45. Yin, R.K.: Case Study Research: Design and Methods, vol. 5. Sage, Thousand Oaks (2009)
46. Yin, R.K., Heald, K.A.: Using the case survey method to analyze policy studies. Adm. Sci. Quart. **20**(3), 371–381 (1975). http://www.jstor.org/stable/2391997
47. Zhang, H., Huang, X., Zhou, X., Huang, H., Babar, M.A.: Ethnographic research in software engineering: A critical review and checklist. In: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Association for Computing Machinery, New York, NY, ESEC/FSE 2019, pp. 659–670 (2019). https://doi.org/10.1145/3338906.3338976

Teaching Systematic Literature Reviews: Strategies and Best Practices

Marcela Genero and Mario Piattini

Abstract Systematic literature reviews have become an important method in software engineering in order to obtain new findings and propose innovative ideas for future research. In this chapter, we share strategies and best practices on teaching systematic literature reviews.

1 Introduction

A systematic literature review (SLR) is a means of identifying, analyzing, and interpreting all available evidence to answer specific research questions in an unbiased and repeatable manner [29]. One of the most significant motivations for conducting SLRs is to obtain new findings and propose innovative ideas for future research [68]. SLRs primarily differ from traditional literature reviews in that they are formally planned and executed in a systematic and methodical manner, thereby reducing researcher bias and increasing repeatability, consistency, and transparency [32, 64]. Furthermore, the process of identifying, evaluating, and synthesizing all pertinent evidence on a given research topic enables an SLR to yield more reliable conclusions than would be possible through the analysis of any individual primary study.

Other types of secondary studies can be classified according to the type of analysis or synthesis made of the primary studies and the sources from which the primary studies are sought: Systematic Literature Mapping or Systematic Mapping Study (SMS) (aka Scoping Review), Grey Literature Mapping (GLM), Grey Literature Review (GLR), Multivocal Literature Mapping (MLM), and Multivocal Literature Review (MLR) [22]. More details on different types of systematic reviews are provided on the chapter "Literature Reviews in Software Engineering Research" by Baltes and Ralph.

M. Genero (✉) · M. Piattini
Department of Technologies and Information Systems, University of Castilla-La Mancha, Ciudad Real, Spain
e-mail: marcela.genero@uclm.es; mario.piattini@uclm.es

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2024

D. Mendez et al. (eds.), *Handbook on Teaching Empirical Software Engineering*,
https://doi.org/10.1007/978-3-031-71769-7_20

The integration of SLR training into the educational programs of PhD and master's students equips them with valuable skills that enhance the quality and impact of their research while preparing them to become more effective and critical researchers in their respective fields.

Failing to perform an SLR can result in a number of significant issues. Firstly, researchers risk duplicating existing studies, thus wasting time and resources. Secondly, without a comprehensive understanding of the current state of knowledge, it is challenging to identify gaps and opportunities for research, which can limit the relevance and impact of new studies. Thirdly, a lack of familiarity with existing literature can result in the misinterpretation of results and erroneous conclusions, negatively affecting the credibility and quality of the research work. Consequently, it is vital to educate students on how to conduct SLRs, not only to enhance their academic development but also to guarantee the integrity and advancement of knowledge within their field of study. This is why it is of paramount importance that academic contexts teach effective SLR that benefits both doctoral and master's students.

While learning to conduct SLRs has many benefits for students (see Sect. 3), it is true that conducting an SLR is a challenging and particularly time-consuming task. A review of existing literature [9, 28, 37, 54] and our own experience reveals formulating research questions, defining the search string, extracting data, assessing quality, and selecting papers as the most challenging aspects of the SLR process, while the most time-consuming aspects of the SLR process are extracting data, selecting papers, searching databases, and assessing quality. In addition to these challenges, there are at least two shortcomings: (1) the lack of tool support, especially for activities like searching databases, selecting papers, and extracting data and (2) the lack of didactic materials. It is well documented that there is a wealth of scientific literature on SLR addressing theoretical issues, as well as numerous articles published on SLR on a variety of topics. However, there is a lack of didactic material or consolidated recommendations on how to address the aforementioned aspects that present difficulties.

In this chapter, we will provide guidance for teachers on how to teach SLR, with a particular focus on the more challenging aspects. We will offer examples, recommendations, tools, support materials, and other resources to assist teachers in their efforts.

It is important to note that the focus of this chapter is on how to teach SLRs, not on how to conduct them. These two approaches are distinct, though the boundaries between them are not always clear. The content of this chapter is enhanced by the inclusion of slides and exercises, which are accessible via the following link: https://zenodo.org/doi/10.5281/zenodo.11544897. This material encompasses all the content covered in the course, offering a comprehensive explanation of each element in the SLR process.

This chapter is organized as follows: Sect. 2 discusses the motivation and rationale behind teaching SLRs, emphasizing their significance particularly to doctoral and master's students. Section 3 outlines the primary benefits of conducting SLRs within the academic and research contexts, also pointing out some drawbacks.

In Sect. 4, we examine the most effective teaching methods for imparting knowledge about SLRs. Section 5 offers a comprehensive teaching plan for SLRs, including learning objectives, teaching units, schedule, and assessment methods. Section 6 provides examples and recommendations for each of the units of the syllabus presented in the previous section. Section 7 offers practical advice for both educators and students on conducting SLRs, while Sect. 8 presents the main tools for performing and teaching SLRs. Furthermore, Sect. 9 provides recommendations for supplementary reading materials that can enhance the learning and practical application of SLRs. The chapter concludes with a summary of the key points discussed and an analysis of the impact of teaching SLRs, as well as potential future developments in this educational field.

2 Motivation and Justification

Every student facing the challenge of writing a master's or doctoral thesis must, to a greater or lesser extent, prepare a "state of the art" on the topic they wish to address. To do this, students can carry out a systematic review, so that it will provide them with the basis from which to develop their thesis proposals.

Conducting an SLR can also be highly beneficial for PhD students, as it enables them to produce the "state of the art" necessary for applying to research projects. This is especially important when they wish to develop their own research line.

Of great interest to students and a potential time-saving strategy in thesis development is to focus on conducting systematic reviews of existing systematic reviews within a given field, such as tertiary studies. This is a viable approach when there are sufficient high-quality systematic reviews on the subject under investigation. A tertiary review can be conducted in accordance with the established process for SLRs, with the advantage that fewer resources are required. Methodological aspects of tertiary reviews have also been the subject of interest, including the assessment of the quality of SLRs [11] and the definition of inclusion and exclusion criteria in tertiary reviews [12].

The aforementioned points illustrate the necessity for master's and doctoral students, as well as researchers in general, to engage in some form of systematic literature review (SLR). Moreover, conducting SLRs enables students to cultivate a spectrum of essential competencies that are vital for their academic and professional advancement:

- Critical thinking and analytical skills: Conducting SLRs requires students to critically evaluate large amounts of literature, discerning the quality of research, identifying biases, and understanding different perspectives within the field.
- Research skills: Students enhance their ability to conduct thorough research, including the formulation of research questions, development of search strategies, and efficient use of various databases and sources.

- Data management skills: Managing, organizing, and synthesizing data from a wide range of sources are of paramount importance when conducting SLRs. Students learn how to handle complex information effectively.
- Problem-solving skills: Dealing with issues such as limited data, conflicting information, or gaps in research teaches students how to navigate and address complex problems.
- Attention to details: The process of meticulously screening studies for inclusion and extracting data demands a high level of attention to detail.
- Writing and communication skills: Writing a systematic review involves structuring a coherent narrative that summarizes and discusses the collected data, enhancing students' academic writing and communication abilities.
- Project management skills: Completing a systematic review requires effective planning, time management, and adherence to deadlines, akin to managing a small project.
- Collaborative skills: Often SLRs are conducted as part of a team, helping students improve their ability to work collaboratively, coordinate with others, and integrate feedback.
- Knowledge acquisition: Through the extensive exposure to existing research, students can deepen their knowledge in specific subject areas, becoming more proficient and knowledgeable in their fields.
- Scholarly publishing experience: By preparing a systematic review for publication, students gain experience in the publishing process, understanding the requirements and standards of scholarly communication.

These skills not only contribute to their immediate research tasks but also prepare them for future academic challenges and professional opportunities.

3 Benefits and Drawbacks of SLRs in the Academic and Research Domain

SLRs offer a range of benefits that are particularly valuable in academic and professional contexts:

- Comprehensive overview: SLRs provide a thorough and exhaustive summary of existing research on a specific topic. This comprehensive approach helps capture the breadth and depth of knowledge, thus ensuring that all relevant studies are considered.
- Reduction of bias: Unlike narrative reviews, SLRs follow a strict, transparent methodology that includes predefined criteria for selecting and evaluating research. This structured approach reduces bias, making the findings more reliable and reproducible.
- Evidence synthesis: SLRs synthesize findings from multiple studies, thereby confirming results or identifying inconsistencies across different research projects.

This synthesis is key to laying a solid foundation of evidence that can inform decision-making and policy formulation.

- Identification of research gaps: By systematically analyzing existing literature, SLRs help identify gaps where further research is needed. This is essential for advancing knowledge in a field and ensuring that future research efforts are directed toward areas of unmet need.
- Support for policy and practice: The evidence collected and synthesized through SLRs can provide a strong foundation for the development of guidelines, policies, and practices. This is of particular importance in fields such as healthcare, education, and environmental policy, where evidence-based decisions can have a significant impact.
- Enhancement of research efficiency: By summarizing what is already known about a topic, SLRs can prevent duplication of research efforts and guide future studies to build on existing knowledge rather than repeating prior work.
- Facilitation of theory development: The findings from SLRs can help in theory development by integrating empirical evidence and identifying patterns or theoretical gaps in the literature. This is crucial for the evolution of scientific disciplines.
- Educational value: For students and new researchers, conducting an SLR is an excellent training exercise in critical thinking, academic writing, and research methodology. It teaches valuable skills such as data collection, analysis, and systematic synthesis of information.
- Credibility and rigor: Publications based on SLRs are generally viewed as credible due to the rigorous methodological framework. This enhances the reputation of the research and can increase the likelihood of publication in prestigious journals.

Overall, SLRs play a pivotal role in many fields by consolidating knowledge, reducing bias, and guiding future research and policy directions, thereby contributing significantly to the advancement of science and informed decision-making. However, there are also some drawbacks, including the following:

- Managing large volumes of information: SLRs often involve collecting and analyzing extensive amounts of data from multiple academic sources, which can be overwhelming and increase the likelihood of errors in study selection and analysis.
- Updating: The field of study may advance rapidly, rendering an SLR outdated shortly after its publication. Keeping the review up to date can require continuous and repeated efforts.
- Required skills: Conducting requires a variety of skills, including knowledge in search methodologies and data synthesis, as well as competencies in managing specific reference tools and statistical analysis software.
- Publication and selection bias: SLRs are susceptible to biases, such as publication bias, where studies with significant results are more likely to be published. Identifying and mitigating these biases is crucial but challenging.

- Consistency and reproducibility: Maintaining consistency in selection criteria and analysis methods throughout the project can be difficult, especially if the review team is large or the project spans a long period.
- Interpretation and synthesis of results: Different studies may use varied methodologies, which can complicate the synthesis and comparison of results. Determining how to integrate and present these findings coherently can be a considerable challenge.
- Limitations in generalizing results: SLRs are limited by the characteristics of the included studies. If the studies are very heterogeneous or focused on specific populations, generalizing the results can be problematic.
- Methodological compromises: In some instances, researchers may be compelled to limit the extent and comprehensiveness of the review because of time and resource constraints.
- Usefulness of SLRs: The usefulness of SLRs is sometimes questioned as it could be argued that they are more of an "end" than a "means" of improving the knowledge of software engineering.

Awareness of the drawbacks will not only assist readers in comprehending the inherent challenges of SLRs but will also encourage them to adopt a more critical and realistic approach to addressing these challenges.

In Sects. 6.3, 6.4, and 6.5, where the process of teaching how to conduct SLRs is explained, some suggestions are presented to help mitigate most of these drawbacks.

4 Specific Teaching Methods

Below, we detail some teaching methods that, according to our experience, are suitable for teaching how to conduct SLRs. The relation of which teaching methods are used in each thematic unit is reflected in Table 1, which shows the teaching plan.

- Lecture-based approach. A lecture-based approach is a traditional teaching method, supported by slides, recommended readings, and other instructional materials, where the instructor delivers a structured presentation on a specific topic to the students. In the context of teaching how to conduct SLRs, this approach involves the instructor providing comprehensive explanations, theoretical foundations, and step-by-step guidelines on the SLR process. It is essential to accompany this approach with relevant examples that illustrate the concepts or topics covered. This approach is used in most of the thematic units, except in Units 6 and 7, which are more focused on practical exercises. The lecture-based approach can be complemented with interactive discussions to foster deeper understanding and practical application of the concepts taught.
- Hands-on workshops. Hands-on workshops are interactive teaching sessions where participants actively engage in practical exercises and activities related to the course material. In the context of teaching how to conduct SLRs, hands-on workshops provide students with the opportunity to apply the theoretical

Table 1 Teaching plan

Thematic units	Main teaching methods	Duration
1. Introduction to research methods and SLRs	Lecture-based	2–5 hours
	Interaction discussion	
2. Scientific repositories and bibliography management systems	Lecture-based	2–3 hours
	Hands-on workshop	
	Group project	
3. Planning an SLR	Lecture-based	2–3 hours
	Case-based learning	
	Interactive discussion	
4. Conducting an SLR	Lecture-based	2–3 hours
	Case-based learning	
	Interactive discussion	
5. Reporting an SLR	Lecture-based	2–3 hours
	Case-based learning	
	Interactive discussion	
6. Reviewing and assessing an SLR	Hands-on workshop	2–3 hours
	Case-based learning	
	Peer-review	
	Interactive discussion	
7. Recommendations for developing SLRs	Lecture-based	2–3 hours
	Interactive discussion	
8. Creating a protocol for an SLR	Hands-on workshop	2–3 hours
	Feedback and review	
	Group project	
	Peer-review	

Table 2 Evaluation activities

Activity	Description	Weight
Ex-1	Performing searches in scientific repositories and using bibliography management systems	10%
Ex-2	Review and assess an SLR	20%
Ex-3	Create a protocol of an SLR	30%
Ex-4	Present the protocol an SLR	10%
Multiple-choice test	A test with questions to evaluate if the student has assimilated the most important concepts	30%

knowledge they have learned in a practical, real-world setting. As shown in Table 1, this type of workshop is used in Units 2, 6, and 8, where students must complete practical exercises, such as using scientific repositories, bibliography management systems, reviewing and assessing an existing SLR, or creating their own SLR protocol (Ex-1, Ex-2, Ex-3, and Ex-4 in Table 2).

- Case-based learning. Case-based learning (CBL) is an educational approach where students learn through the analysis and discussion of real or simulated cases. In the context of a course on how to conduct SLRs, CBL can be particularly effective because it allows students to engage deeply with the practical aspects of the research process. By working with real-world examples, students gain hands-on experience in identifying research questions, evaluating the quality of studies, and synthesizing evidence. This method also encourages critical thinking and problem-solving, as students must navigate the complexities and challenges that arise in actual research scenarios. Additionally, CBL fosters collaborative learning, as students discuss and debate their findings and approaches, leading to a richer understanding of the material. Specifically, in the thematic units where the SLR process is explained though a lecture-based approach (Units 3, 4, and 5), real cases (i.e., examples of existing SLRs) are presented. Through interactive discussions, students critically analyze each case. Additionally, in Unit 6, students will encounter real cases, namely, existing SLRs, which they must review and critique as well as assess their quality.
- Group projects. Group projects in SLR training foster a collaborative learning environment where students can develop essential group competencies such as communication, conflict resolution, and shared decision-making. Many activities can be structured for group participation, enriching perspectives and integrating diverse viewpoints. These activities mirror real-world research scenarios where multidisciplinary teams collaborate, preparing students for professional research efforts. Furthermore, group work can alleviate individual workload and provide a support system, thereby fostering deeper understanding and a more enjoyable learning experience. In our teaching plan, several activities could be designed for group participation. For instance, we propose a group activity where students work in pairs to complete exercises related to the use of scientific repositories and bibliography management systems (Ex-1 in Table 2), allowing them to experience this collaborative approach firsthand. Additionally, the peer-review activity in Unit 8 is conducted in groups of two students.
- Interactive discussions. Discussions typically arise when reviewing specific aspects of an SLR. In addition to evaluating research questions or inclusion/exclusion criteria, the criteria for assessing the quality of primary studies are often subjects of interactive discussions, among other topics. In most thematic units, interactive discussions are conducted to encourage student participation.
- Feedback and review sessions. A series of sessions is proposed in Unit 8, during which feedback will be provided and students will be guided (in a coaching mode) so that they can engage in critical thinking and reflection on the elements to be defined in their own review protocol. It is recommended that each student undertake an independent experiment on their own SLR, with a specific focus on the research topic they intend to address in their master's thesis or doctoral dissertation.
- Peer review. In this context, students are required to develop an SLR protocol (Ex-3 in Table 2) in Unit 8. Subsequently, they exchange protocols in pairs to provide feedback to each other. This peer review process allows students to

critically assess and offer constructive feedback on their peers' work. It enhances their understanding of SLR process and cultivates skills in collaborative learning and effective communication. This approach fosters a supportive academic environment where students benefit from diverse perspectives and constructive criticism, ultimately improving the quality and depth of their research endeavors.

5 Teaching Plan

To ensure the teaching plan is well understood and consistent with the learning objectives, it seems appropriate to mention that we focus on the "classic" process proposed by [29], which consists of three main phases, each containing several stages, as depicted in Fig. 1.

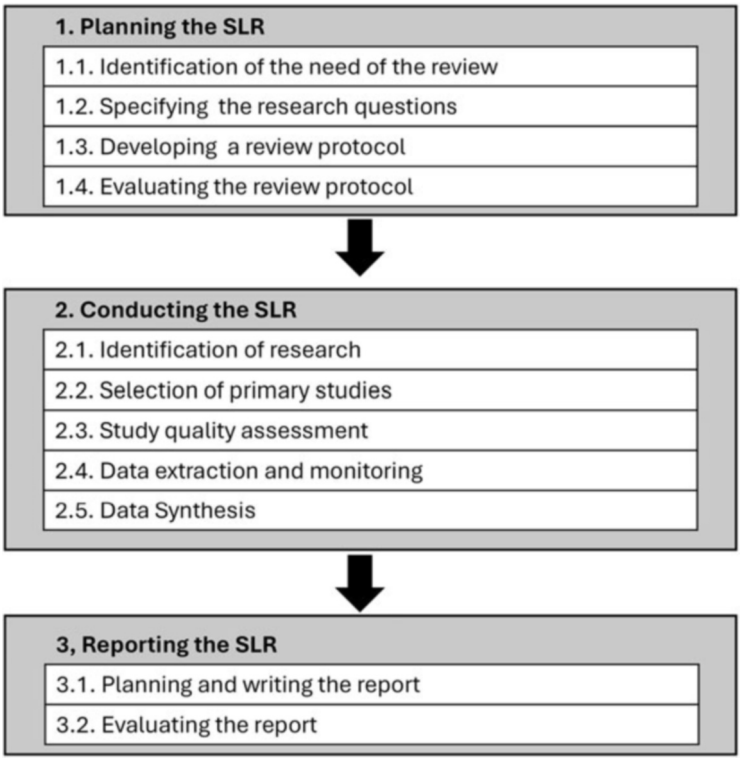


Fig. 1 SLR process

5.1 Teaching Units

To design the teaching plan, we start by presenting in Table 1 the thematic units and the appropriate teaching methods for each unit, as well as the estimated time needed to develop each unit (excluding students' autonomous work beyond each class).

Below, we will describe the learning objectives (LO) for each of the thematic units outlined in Table 1.

Unit 1. Introduction to research methods and SLRs
Students should:

- LO1.1. Be able to identify the different types of research studies (primary, secondary, and tertiary)
- LO1.2. Be familiar with different types of research methods used in primary studies (experiment, survey, case study, action research, etc.)
- LO1.3. Be able to understand why and when an SLR should be undertaken
- LO1.4. Be aware of the benefits and drawbacks of SLRs
- LO1.5. Be familiar with the different types of SLRs

Unit 2. Scientific repositories and bibliography management systems
Students should:

- LO2.1. Be familiar with the main scientific repositories
- LO2.2. Be familiar with the steps involved in searching in the main repositories
- LO2.3. Be able to use some bibliography management systems

Unit 3. Planning an SLR
Students should:

- LO3.1. Be able to identify the need of the review
- LO3.2. Know and understand the elements that compose a review protocol
- LO3.3. Be able to formulate the research questions
- LO.3.4. Be aware of different search strategies (automatic, manual, hybrid, etc.)
- LO.3.5. Be able to define the search string, the search sources, and the search period and where to search within each selected primary study
- LO.3.6. Be able to define the study selection criteria (inclusion/exclusion)
- LO3.7. Be able to define the study selection procedure
- LO.3.8. Be able to design the data extraction form including (a classification scheme)
- LO.3.9. Know different data synthesis methods
- LO.310. Be able to define a data synthesis procedure
- LO.3.11. Be able to define the quality assessment instrument
- LO.3.12. Be able to define a schedule for developing an SLR
- LO.3.13. Be aware of the need to ask an expert to review the protocol

Unit 4. Conducting an SLR
Students should:

- LO.4.1. Know the different steps for conducting an SLR

- LO4.2. Be able to identify the primary studies applying the search strategy
- LO4.3. Be able to select the primary studies applying the study selection procedure
- LO4.4. Be able to perform the study quality assessment on the selected primary studies
- LO4.5. Be able to do the data extraction
- LO4.6. Be able to carry out the data synthesis
- LO4.7. Be able to document all incidents and decisions that occurred during execution of the review

Unit 5. Reporting an SLR
Students should:

- LO5.1. Know the structure and content of a report of an SLR
- LO5.2. Be familiar with the structure and content of a report of an SLR
- LO5.3. Be aware of where and how to publish an SLR
- LO5.4. Be aware of open science practices and their impact on the reproducibility of SLRs
- LO5.5. Be aware of the need to have the full report reviewed by an expert, if possible, before submitting it to a conference or journal

Unit 6. Review and assess an SLR
Students should:

- LO6.1. Be able to identify the elements of the review protocol in and existing SLR
- LO6.2. Be able to offer constructive criticism on an existing SLR
- LO6.3. Be able to explain the entire SLR process of an existing SLR
- LO6.4. Assess the quality of an SLR

Unit 7. Recommendations for developing an SLR
Students should:

- LO7.1. Know lessons learned or tips related to the three stages of the SLR process
- LO7.2. Know the more important ethical aspects when performing and reporting an SLR
- LO.7.3. Be aware of the issues that can pose threats to the validity of an SLR
- LO7.4. Be familiar with the main tools for performing an SLR
- LO7.5. Reflect on the usefulness of an SLR
- LO7.6. Be aware on the most relevant an up-to-date literature about SLRs

Unit 8. Create a protocol for an SLR
Students should:

- LO8.1. Be able to justify the need of a new SLR
- LO8.2. Create a protocol for an SLR
- LO8.3. Be able to formulate the right search questions and explain the rationale behind them
- LO8.4. Be able to argue each of the elements of the defined protocol

- LO8.5. Be able to present and explain the entire process of an SLR
- LO8.6. Be able to argue and debate on any decisions taken along the SLR process

5.2 Duration and Schedule

The durations provided for each unit are only an estimation based on our experience. They may vary significantly based on the availability of time, the number of students, and their previous knowledge. Even within a PhD course, the length of time devoted to SLRs can vary considerably, from a few hours to more than a dozen, and in a very intensive and compressed format to a more extensive format spread over several weeks. For example, if students have already covered the content of Unit 2, this could be removed from the syllabus. Also, as discussed, in our courses, we ask them to submit the protocol for an SLR, because we consider that within the time frame of the course and with the knowledge acquired, completing a full SLR would be excessive. This is because it requires the students to have a more solid understanding of the concepts and greater skill. This skill can be developed by reading several SLRs. The time allocated for the introduction could also be reduced if they have already taken another course on research methods.

Furthermore, the number of attendees to the course must be considered when adjusting the material. We have taught these teachings in small courses of 5/8 people to massive courses of more than a hundred attendees.

5.3 Learning Assessment

When teaching SLRs is included as part of an official doctoral or other course, evaluation is essential. Table 2 shows how evaluation can be conducted using different activities, such as hands-on exercises and tests. It also shows the weight that each activity has on the final grade, which we have established based on our experience. For example, the weights could be distributed differently in the absence of a final test.

It is not common to have enough time in courses for students to perform an SLR. Therefore, we usually ask them to only define the protocol for their SLR. Additional feedback sessions could be distributed throughout the course with more time available. Feedback is essential, particularly when students are developing a protocol for their first SLR. As SLR course instructors, we provide feedback that is primarily methodological in nature. However, this should always be complemented by the thesis supervisor, who has the deepest knowledge of the specific domain of the SLR.

6 Examples and Recommendations

In this section, we provide examples and recommendations for each unit of the syllabus outlined in the previous sections. Additionally, detailed information is available on the slides of each unit.

6.1 Unit 1. Introduction to Research Methods and SLRs

This introductory unit explains the types of research studies: primary, secondary, and tertiary. Regarding primary studies, the main research methods are experiments, case studies, surveys, and action research. We believe it is important to explain the main characteristics of these methods because when conducting an SLR or trying to understand an existing one, we will need to understand the selected primary studies.

To demonstrate the significant growth of secondary and tertiary studies, we conduct searches in Scopus, one for secondary studies and one for tertiary studies, and present graphs that evidence such growth. Regarding tertiary studies, we highlight the topics on which tertiary studies exist. We use Scopus because of its flexibility in defining search strings and because it indexes multiple digital libraries.

For example, we use the following search string for secondary studies:

"systematic literature review" OR "systematic mapping" OR "mapping study" OR "systematic review"

restrict the search to the abstract-keyword and title, to the computer science area and written in English.

We do something similar for searching tertiary studies.

We explain why and when an SLR must be undertaken and the benefits and drawbacks thereof (see Sect. 3). We also present the different types of secondary studies, classified according to the type of analysis/synthesis made of the primary studies, and the sources from which the primary studies are sought. We place particular emphasis on the difference between SLR and SMS, as these terms are sometimes confused.

For this unit, we primarily recommend the following bibliography: [22, 31, 35, 63]. For Spanish speakers, we recommend [24].

To illustrate all the stages of SLRs when teaching them, we use the following examples: [18, 23, 57]. Additionally, we provide students with several papers containing examples of SLRs and SMSs, such as those by [4, 21, 25, 26, 46, 48, 56]. We also recommend that students consult the catalog of SLRs for educational and practical purposes provided in [33], which includes examples from various knowledge areas.

6.2 Unit 2. Scientific Repositories and Bibliography Management Systems

In this unit, we present the main repositories used in software engineering: Scopus, IEEE Xplore, ACM Digital Library, ScienceDirect, and Springer. Due to its flexibility, we delve deeper into Scopus and conduct hands-on exercises to explain its most relevant functionalities, such as advanced search options, results filtering, and citations tracking. Understanding how to efficiently use these repositories is crucial for conducting thorough literature reviews and staying updated with the latest research in the field.

We also introduce different bibliography management systems, such as Mendeley, EndNote, and Zotero. These tools are essential for organizing references, creating bibliographies, and sharing research with peers. We provide a more in-depth explanation of Mendeley, covering its key features such as importing references, organizing them into folders, generating citations in various formats, and collaborating with other researchers through shared libraries. Hands-on exercises are used to illustrate its full functionality, ensuring that students gain proficiency in using the software to streamline their research workflow.

Additionally, we discuss best practices for managing and organizing research data, emphasizing the importance of consistency and accuracy in citation management. We also explore how integration with word processors and other tools can enhance productivity and reduce the likelihood of errors in manuscript preparation.

The hands-on exercises of this unit correspond to Ex-1 in Table 2 and are available at https://zenodo.org/doi/10.5281/zenodo.11544897.

6.3 Unit 3. Planning an SLR

This unit covers the four steps of planning an SLR as shown in Fig. 1. Each step is described in more detail below.

The first step is to establish or explain the rationale behind the need to develop new SLRs. Therefore, it is advisable to first check whether there are any existing SLRs on the topic we are researching and, if so, decide whether it is necessary to update them. For guidance on updating SLRs, we recommend reading the guidelines provided in [41, 65].

Specifying research questions is a critical part of planning an SLR, and the factors that motivate the questions should be fully explained. It is common practice to formulate a clear and focused main research question and to explain the rationale behind it. If necessary, additional specific research questions needed to fully address the main research questions can also be identified and explained. Regardless of the objectives of an SLR, there are some generic research questions that are advisable to consider in an SLR, as they can help provide an overview and provide relevant information on existing publications [36]:

- How many publications on (the topic of interest) have been published?
- How many publications on (the topic of interest) have been published over the years?
- What is the scientific maturity of the set of publications?
- What contributions do the publications propose?
- What are the main trends observable in the set of publications?
- What new approaches exist on the topic of interest?

The definition of the main issues of the search process, primary study selection process, data extraction process, and data synthesis process are described on the template in Table 4.

For justifying automatic or manual searches, there are good examples in [33, 43, 66] present studies that demonstrate the effectiveness of hybrid searches, which combine database searches with snowballing procedures. The snowballing procedures, which includes forward and backward snowballing, are detailed in [62].

Once the scope of the study and the research questions has been established, researchers should consider the appropriate search strings, which also depend on the domain being investigated. Depending on the accuracy of the search strings, searches may yield inappropriate results, too much overhead, or simply an incomplete set of results. Therefore, search strings must be defined carefully and always tested before the actual search. Defining the search string usually requires several iterations until the final string is defined; and it is advisable to define it in English, since this is the international language used in research, although, in very justified cases, it may be necessary to include another language.

In order to formulate the search string, it is advisable to contact experts in the topic of interest and ask them to provide relevant articles in order to gather ideas about the terms that could be included in the search string [29].

The search string should be defined following the guidelines provided by [7]. Initially, it is advisable to formulate several search strings and apply them to a chosen search source. By reviewing the titles and abstracts, we can assess whether the articles retrieved align with the objectives of the Systematic Literature Review (SLR). If an excessively large number of articles is retrieved, further refinement of the search string may be necessary to ensure more targeted results.

The inclusion and exclusion criteria for primary studies should be defined with the goal of identifying primary studies that provide evidence related to the research questions. To reduce the likelihood of bias, the selection criteria should be established during the protocol definition, although they may be refined during the search process. These should also be specified within the exclusion criteria to exclude duplicate studies, in case two studies are published in different articles; typically, the less comprehensive one is excluded. The following are examples of inclusion (I) and exclusion (E) criteria:

- I. The title, list of keywords, and abstract explicitly indicate that the article is related to the researched topic.
- I. The article presents contributions related to the researched topic [list the topics].

- E. The article is not in English (nor in any other language of interest).
- E. The article is not within the domain [indicate the name(s) of the domain(s)].
- E. The article is a summary of a tutorial, workshop, or poster.
- E. The article relates to the topic only in its related work section.
- E. The article appears multiple times in the results set.

While there is no standard process to perform the selection of primary studies, we propose the following steps that the study selection process should include:

- Step 1. Conduct the searches. Once the searches have been completed, it is advisable to record in a table the articles found, their abstracts, and the source from which each article was found.
- Step 2. Remove duplicates. The table created in the previous step should be checked for duplicates; it is advisable to sort the table by article title.
- Step 3. Apply inclusion/exclusion criteria by reading the abstract. In this step, you will decide which articles to include or exclude by applying the defined criteria based solely on the abstract. In case of doubt, proceed to the next step.
- Step 4. Apply inclusion/exclusion criteria by reading the full text. In this step, some articles previously included may be excluded upon full text review.
- Step 5. Quality assessment. This step is applied in the selection process when quality criteria have been defined with the aim of excluding those that do not meet a certain quality criterion. Otherwise, in the previous step, we will have already obtained the set of articles that will later be analyzed and synthesized to answer the formulated research questions.

This selection process should be accompanied by a table like the one shown in Table 3.

In addition to defining inclusion/exclusion criteria, it is important to assess the quality of primary studies to:

- Provide even more detailed inclusion/exclusion criteria
- Investigate whether differences in quality provide a justification for explaining differences in study outcomes
- Weight the importance of individual studies when synthesizing results

Table 3 Example of a table to record the selection process

Sources	Remove duplicates	Apply inclusion/exclusion criteria reading the abstract	Quality assessment	Number of selected articles
IEEE XPLORE				
ACM DIGITAL LIBRARY				
SCOPUS				
...				
Number of selected articles				

- Guide the interpretation of results and determine the strength or validity of inferences made
- Provide recommendations for future research

The assessment of the quality of primary studies in an SLR is usually done through checklists such as the one provided by [17]. In the case of SMSs, the assessment of the quality of primary studies is not mandatory [29, 52], although there are numerous examples of SMSs where it is presented. In these cases, the criteria for evaluating quality are very diverse, as can be seen in [20, 21, 44, 49, 53, 55]. In many cases, the justification of quality is limited to stating that they are publications subject to a rigorous review process (as in the case of indexed journals or prestigious conferences with peer review).

A data extraction form must be designed to extract all the relevant information needed to answer the research questions posed. In addition, the definition of a classification scheme for the selected primary studies is often essential, especially in SMSs.

In [52], a distinction is made between classification schemes that are general or independent of the research topic and those that are dependent on the topic. The topic-independent classification scheme should include at least three categories: the forum where the work is published (detailing journals, conferences, etc., where the articles were published), the type of research, and the research method used. For these last two aspects, it is recommended to use the classification proposed by [61], which identifies the following categories for the type of study: evaluation, validation, solution proposal, philosophical article, experience report, and opinion article. It is important to note a common confusion in the classification between "validation" and "evaluation." Validation is conducted in a controlled environment, such as a laboratory, while evaluation is performed in a real industrial context. Furthermore, even if a new solution is justified for implementation in practice, it remains a "proposed solution" until empirically evaluated. The selection of the research method should be consistent with the classification of the type of research, as illustrated in Fig. 2.

The classification scheme related to the specific topic may arise from the study being conducted or may be based on existing literature. When possible, it is useful to consider an existing classification as a baseline, as this supports comparability between mapping studies. Before starting the classification, we suggest consulting experts in the field to identify existing classification schemes. [49] provide an example of such a classification scheme.

In the data extraction form, it is advisable to include (1) the metadata of the publication, such as title, authors, year, type of publication, etc.; (2) specific fields that serve to answer the research questions or to classify the articles according to the defined classification scheme; and (3) fields that include the items defined in the checklist for assessing quality.

Data extraction and classification of primary studies are carried out by reading the full text. It should be specified who will be responsible for performing the data extraction and who will ensure that the data extraction is done correctly.

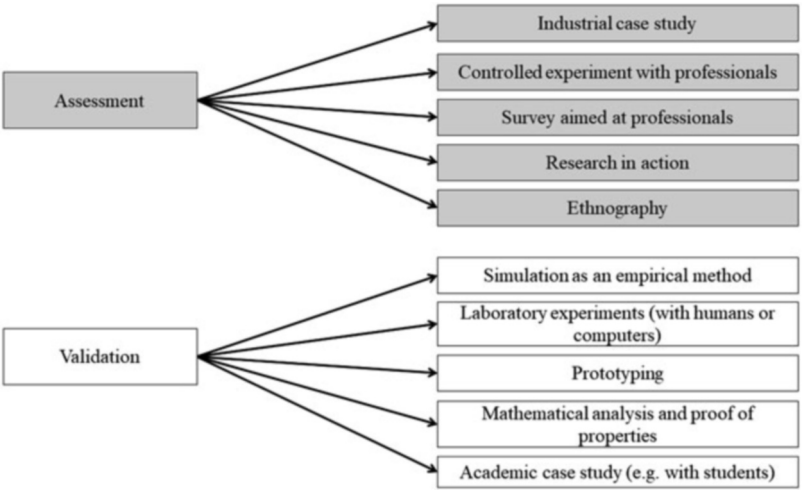


Fig. 2 Classification of research methods

It is advisable that at least one researcher randomly selects some articles and verifies whether they are correctly classified or not. It should also be indicated how discrepancies, for example, in the classification of articles, will be resolved.

Research synthesis is crucial, although it is one of the most labor-intensive tasks and has received the least attention in systematic literature reviews (SLRs) [13]. In an SLR, data synthesis is conducted to address the research questions posed. Synthesis encompasses a family of methods used to synthesize, integrate, combine, and compare the results from different studies on a specific topic or research question [10, 16, 45].

If the primary studies have similar independent and dependent variables, it may be possible to aggregate them through meta-analysis, which uses statistical methods to combine effect sizes. However, in computing, primary studies are often too heterogeneous to allow statistical synthesis, and particularly for qualitative studies or studies using various research methods, different research synthesis methods are needed as presented in [13]. Examples of the application of some synthesis methods can be found in [5, 53] for narrative synthesis; [18] for meta-ethnography; [4, 58] for thematic analysis; [27] for case survey; and [30, 59] for comparative analysis.

It is also advisable to create a table with the timeline for each activity and try to adhere to it to avoid delays. However, it is true that without knowing the volume of articles to review upfront, the estimation made in the initial stages of the review may not be entirely accurate. In such a case, it will be necessary to adjust the planning.

Finally, as part of the protocol definition, it is advisable to have it evaluated by experts. It would be appropriate to contact a group of independent researchers to evaluate the protocol. These same experts can then validate the SLR report.

6.4 Unit 4. Conducting an SLR

This activity puts into practice everything that was previously planned in the protocol, and the final results that will answer the research questions are obtained. Additionally, it is essential to document all incidents and decisions that occurred during the tasks carried out during the execution of the review. This will make the SLR replicable and ensure that all decisions made are available to external reviewers and those who wish to use the results obtained. When conducting an SLR the steps outlined in Fig. 1 must be performed.

The set of publications relevant to answering the research questions is identified by following the search strategy defined in the protocol. Due to the limitations presented by digital library search engines [7], it is often necessary to refine or adapt the search strings. It may also be necessary to refine the search strings, include new sources, or change the search period due to discoveries made during the searches. Therefore, it is important to document the modifications made to the search strategy and save the results using reference management systems such as EndNote, BibTex, etc. In addition to the metadata of each article (title, authors, year, etc.), it is important to save the abstract.

In this task, duplicate articles found in multiple sources must also be detected and eliminated. The selection process should locate the primary studies that provide evidence related to the research questions. This process should also follow what was planned in the protocol according to the criteria and procedure for the selection of studies established. If an article is unavailable during the selection process, we recommend not merely stating that the article was excluded for this reason. Instead, it is advisable to seek the article through the authors' personal Web pages, platforms like ResearchGate (https://www.researchgate.net/), or by contacting the authors directly via email. Alternatively, reaching out to other researchers who may have access to the articles can also be effective.

As a result of this activity, a list of selected primary studies should be obtained, stored in some reference management system, including also the files with the articles in electronic format and the list of studies not included, together with the justification for their exclusion. Once the primary studies are selected, they will be subjected to a quality evaluation process applying the checklist defined in the protocol. As a result of this task, it may be necessary to exclude those articles that do not meet the established threshold to be considered quality studies, or simply the quality is evaluated to reflect the quality of the studies included in the SLR.

During the extraction of relevant data, the data extraction form defined in the protocol will be filled out. It is advisable to have the data extraction validated by another researcher, at least by randomly selecting some studies and attempting to resolve any discrepancies that may exist. As a result of this task, the data extraction forms filled with the information corresponding to each selected primary study will be obtained.

We caution that duplicates may be found in the data extraction process and some studies may need to be excluded. For example, it is common for an experiment

to be published at a conference and then for experiment and its replications to be published as a family of experiments, in a journal, which would require the exclusion of the study originally published at a conference.

Once all the relevant data for each primary study are compiled in the data extraction form, they will be synthesized using the methods established in the protocol to answer the formulated questions. The synthesis is usually accompanied by tables and charts to illustrate the results. Bubble plots like the one shown in Fig. 3 are very effective for synthesizing data by combining several research questions.

It is also advisable to use a bar chart, such as the one in Fig. 4, to show the evolution of the number of papers over the years.

6.5 Unit 5. Reporting an SLR

It is advisable to follow SEGRESS guidelines [35] when reporting SLRs. These guidelines include variants that can be applied to both quantitative systematic reviews and mapping studies, as well as qualitative reviews.

Embracing open science and ensuring reproducibility are critical aspects of conducting and reporting SLRs. Following are a number of open science practices that should be incorporated into the SLR process:
Pre-registration and protocols:

- Pre-register your SLR protocol on platforms designed for a broader range of disciplines than just health, such as the Open Science Framework. This allows you to outline your research question, methodology, and planned analyses transparently, which helps in reducing bias and enhancing reproducibility.
- Publicly share your protocol to invite feedback, foster collaboration, and avoid duplication of efforts. Platforms like GitHub can also be used for sharing protocols and getting contributions from other researchers, which is particularly relevant in software engineering.

Open access tools and resources:

- Use open-access databases and repositories that are most relevant to software engineering. These might include IEEE Xplore, ACM Digital Library, and arXiv for computer science preprints.
- Employ open-source tools for managing your references and data, such as Zotero for reference management. For qualitative and quantitative data analysis, consider using software like R, Python, or open-source frameworks tailored to your analysis needs.

Data sharing:

- Plan to deposit your datasets in accessible, reputable data repositories that cater to a broad range of scientific disciplines, such as Zenodo or Figshare. These

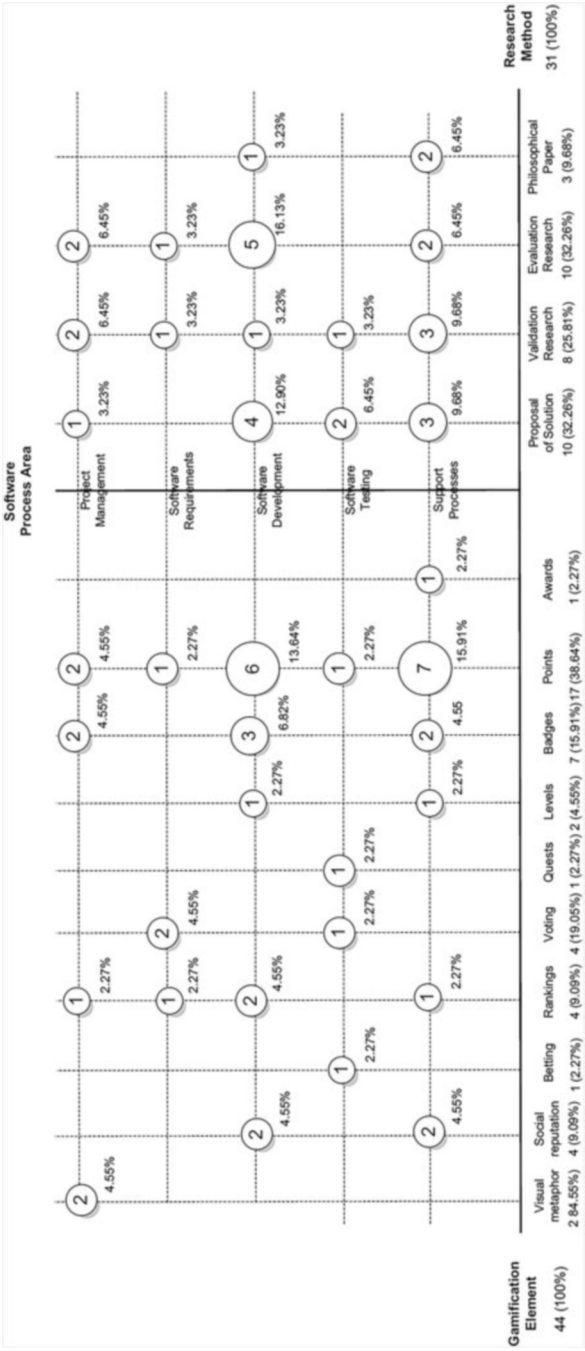


Fig. 3 Summary of the classification by combining dimensions [49]

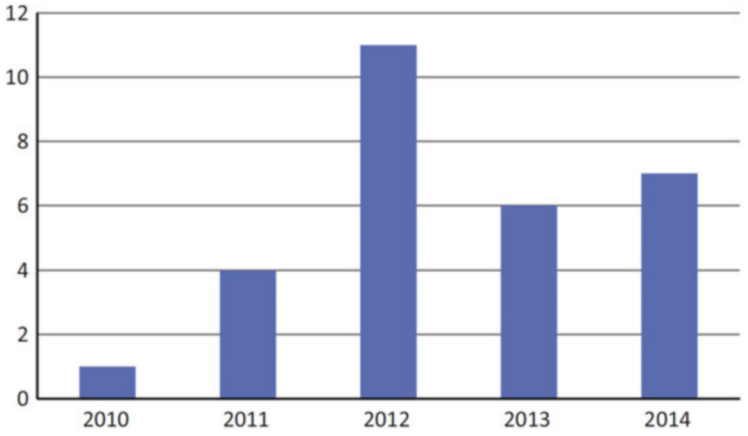


Fig. 4 Distribution of primary studies by year [49]

platforms facilitate data sharing across the scientific community, enhancing the potential for replication and secondary analysis.
- Ensure your data is well prepared for sharing by organizing, documenting, and anonymizing it as necessary. This preparation makes it easier for others to use and understand your data, thereby supporting the principles of open science.

Additional considerations for software engineering:

- Incorporate software and code sharing by using repositories like GitHub or GitLab, where you can host and manage your code, track changes, and enable others to contribute. Sharing your review's computational environment and scripts can be crucial for replicability in software engineering.
- Consider the use of preprints to disseminate preliminary findings quickly. Platforms like arXiv or SSRN (Social Science Research Network) can be utilized to share early versions of your work, reaching your community swiftly and facilitating early feedback.

Adopting these practices ensures that the SLR not only adheres to the principles of open science but is also more aligned with the specific needs and practices of the software engineering research community.

Below are some recommendations for enhancing reproducibility:

Transparent reporting:

- Adhere to guidelines: Utilize standards like PRISMA to ensure thorough and transparent reporting. In software engineering, documenting each phase of your methodology (requirements gathering, selection of sources, criteria for inclusion and exclusion) is crucial.
- Supplementary materials: Include detailed appendices or supplementary sections that outline your search strategy, study selection, data extraction procedures, and

criteria for quality assessment. This enables others to understand and replicate your methods.

Version control:

- Systematic management: Use tools like Git to manage revisions of your documents and code. This is vital in software engineering where changes to code and documentation are frequent and need to be tracked meticulously to reproduce results accurately.
- Platforms for collaboration: Utilize platforms like GitHub or GitLab not just for version control but also for facilitating collaboration and transparency with other researchers.

Reproducible workflows:

- Automation tools: Employ scripting and programming environments such as R or Python. These can automate the execution of complex analyses and ensure that your work is reproducible with the execution of a single script.
- Documentation: Provide detailed documentation of the code, scripts, and any software configurations used. This approach not only facilitates reproducibility but also assists in the debugging and refinement of the tools.

Critical appraisal and quality assessment:

- Standardized tools: Use established tools for critical appraisal to evaluate the quality of the studies reviewed. Clear reporting of these evaluation methods enhances the credibility and reproducibility of your conclusions.
- Transparency in evaluation: Make the criteria and process of study evaluation transparent. This aspect is especially important in software engineering SLRs where technological advancements and trends can affect study relevance and quality.

Collaborative platforms:

- Open science communities: Engage with platforms that promote open science practices like protocols.io, which allows sharing of detailed research protocols, or PubPeer for engaging in post-publication discussions.
- Community feedback and interaction: These platforms facilitate the acquisition of constructive feedback and the establishment of a community of interest around you specific research area. This, in turn, can lead to improvements in your work and to a wider acceptance and application of your findings.

Implementing these practices will not only bolster the reproducibility of your SLR in software engineering but will also enhance its credibility, impact, and usefulness to other researchers and practitioners in the field.

6.6 Unit 6. Review and Assess an SLR

Students can choose an SLR related to their research topic, or if necessary, the teacher may recommend one. Students must identify each element of the review protocol discussed in class within the article.

To search for an SLR of interest, we recommend conducting a search on Scopus using the following search string:

("systematic literature review" or "systematic review" or "systematic mapping" or "mapping study") AND . . .

In the "AND" field, a search string within the specific research topic must be included. For instance, if the topic is the use of gamification in programming, the complete string might appear as follows:

("systematic literature review" OR "systematic review" OR "systematic mapping" OR "mapping study") AND (gamification OR "serious game") AND (programming)

Furthermore, students are requested to evaluate the quality of the selected SLR using the instrument provided in [60]. Subsequently, in the classroom setting, each student presents the protocol of the SLR, and a discussion and feedback session is conducted with the objective of reinforcing the concepts encountered in class regarding each element of the review protocol.

The exercises assigned to the students in this unit correspond to Ex-2 shown in Table 2.

6.7 Unit 7. Recommendations for Developing SLRs

It is our typical practice to issue recommendations at each stage of the SLR process. As an illustration, we may cite those of [9], for example, concerning:
Search strategy:

- Tailor the search strategy for each database.
- Plan to manipulate the search string for each database or source.
- Databases overlap, so define a strategy for identification and elimination of duplicates results.
- Do not underestimate the effort required to combine references exported from different databases or sources.

Identification of primary studies:

- Expert review of inclusion/exclusion criteria.
- Collaboration in reviewing and selecting the papers.
- Strategies for team management and conflict resolution.

Quality assessment:

- Define appropriate quality assessment criteria.

- Apply the criteria to eliminate low-quality studies.
- Have multiple authors independently assess quality.

Data extraction:

- Have an expert review the data extraction form.
- Support data extraction and review by multiple authors.
- Ease recording and analysis of the extracted data.

As previously noted, the utility of SLRs is occasionally called into question. It appears that they serve more as an "end" than a "means" of enhancing the knowledge of software engineering. In this regard, we present the resonance schema, which was developed by [15], for categorizing SLR contributions. This schema can be beneficial for a variety of stakeholders, including reviewers, producers, and consumers. Master and PhD students can be SLR's producers and consumers. As producers of an SLR, they can use the scheme to capture the contributions of their SLRs and as consumers looking for research gaps, which can be easily identified as contributions of interest.

To identify and mitigate potential threats to the validity of an SLR, we recommend the following guidelines provided in [3].

6.8 Unit 8. Create a Protocol for an SLR

To help students create and report a protocol, we propose a template as shown in Table 4, which has been adapted from the one proposed by [33].

In addition, when building the protocol, we recommend applying the knowledge gained from the unit on planning SLRs, as well as the tips discussed in class and detailed in Sect. 7.

We also request that students present their own protocol in class using slides. This will provide the opportunity to share their protocol with all students and to view various examples. The assignments required for this unit correspond to Ex-3 and Ex-4, as shown in Table 2.

7 Tips on Teaching SLRs for Teachers and Students

While SLRs offer numerous advantages, they are also time- and effort-consuming. Consequently, one of the main challenges in conducting SLRs is to strike a balance between methodological rigor and the effort required.

Here are some tips or suggestions for teachers based on our own experience on teaching SLRs for more than 25 years:

- Introduce the purpose and value of SLRs: Start by explaining what SLRs are, why they are important in research, and how they differ from other types of

Table 4 Protocol template, adapted from [33]

Rationale of the SLR	Explain why there is a need for a secondary study on this topic
	If extending or updating previous research on the topic, explain the need for this
Research questions	Specify the main research question being addressed by this study and explain the objective and rationale behind this question
	Specify and explain the objectives of any other research questions that will be addressed
Search process	Specify and justify the search strategy: manual search, automated search, or hybrid
	For automated searches, specify search terms and compounds of these and record results of any prototyping of search strings
	For automated searches, identify resources to be used (specifying the digital libraries and search for manual searches, identify the journals and conferences to be searched)
	Specify the time period to be covered by the review and any reasons for your choice
	Specify in which parts of the articles the search will be conducted, for example, in the title, abstract, and keywords
	Identify any ancillary search procedures, for example, asking leading researchers or research groups accessing their Web sites or checking reference list of primary studies
	Specify how the search process is to be evaluated (e.g., against a known subset of paper, against the results from a previous systematic review, etc.)
Primary study selection process	Identify the inclusion/exclusion criteria for primary studies
	Define how selection will be undertaken (roles of reviewers)
	Define how agreement among reviewers will be evaluated
	Define how any differences between reviewers will be resolved
Study quality assessment process	Specify the quality checklist to be used
	Specify how the checklist will be evaluated (if a new checklist has been developed)
	Define how agreement among data extractors will be evaluated
	Define how any differences between data extractors will be resolved
	Identify the procedures to use for applying the checklist, such as details inclusion/exclusion, partitioning the primary studies during aggregation or meta-analysis, and explaining the results of primary studies
Data extraction process	Design the data extraction form (and test it in a dry run)
	Specify the strategy for extracting and recording the data
	Identify how the data extraction process is to be undertaken and validated, particularly any data that require numerical calculations, or is subjective
Data synthesis process	Specify the form of analysis/synthesis to be used (e.g., narrative, tabulation, meta-analysis, etc.)
	Discuss how the synthesis will be validated
Schedule	Provide time estimates for all the major activities of each of the stages of the SLR process

literature reviews. Highlight their value in identifying consensus, research gaps, and informing practice and policy.
- Teach the process step by step: Break down the SLR process into manageable steps. Typical steps include formulating the research question, defining inclusion and exclusion criteria, systematically searching the literature, selecting studies, extracting data, assessing the quality of the studies, and synthesizing the findings.
- Developing effective research questions: Instruct on how to formulate clear, specific, and feasible research questions.
- Systematic searching and reference management: Teach how to perform systematic searches using different databases and search engines. Discuss the importance of using relevant keywords and search terms. Introduce reference management tools like EndNote, Zotero, or Mendeley to organize sources.
- Critical appraisal of the literature: Teach how to critically assess the quality and relevance of the included studies. You can teach them to use checklists and quality assessment instruments, such as the one proposed in [60].
- Data synthesis and writing: Show how to effectively synthesize extracted data, which can be quantitative (meta-analysis) or qualitative (narrative synthesis). Additionally, teach best practices for writing up the findings of an SLR, following reporting guidelines like SEGRESS [35].
- Using specialized tools: Familiarize students with specialized tools that can facilitate conducting SLRs, such as the ones mentioned in Sect. 8.
- Practice and feedback: Provide opportunities for supervised practice through exercises or small projects and offer constructive feedback. Practical experience is key to developing the necessary skills for conducting SLRs.
- Encourage research ethics: Highlight the importance of academic integrity, transparency, and ethical reporting throughout the SLR process.
- Maintain a learning attitude: Encourage students to stay updated with the latest trends and best practices in conducting SLRs, as it is a rapidly evolving field.
- When teaching SLRs, it's crucial to balance theory with practice, providing students with the tools, skills, and knowledge needed to effectively and ethically conduct SLRs.
- To effectively explain the entire SLRs process, it is advisable to select two or three exemplary SLRs as references for the entire course. These examples can serve as detailed guides for illustrating the key elements of the SLR protocol and each step of the process in depth.

Furthermore, we think it is appropriate to add some tips that the teacher can transmit to their students, such as the following:
Tips related to the Planning phase:

- Start reading several SLRs: For novice students, understanding the SSLR process is complex. When embarking on a new SLR, it is recommended that students familiarize themselves with several examples suggested by their supervisor.
- Understand the purpose: It is important to understand the rationale behind the SLR approach. Awareness of their function in collating and integrating

all pertinent studies on a given subject can provide motivation throughout the intricacies of the process.
- Start with clear objectives: Formulate a clear and focused main research question, explaining its rationale. If necessary, specify and explain additional specific research questions that are required to comprehensively address the main research question.
- Develop a protocol: Before starting your review, plan your process. This includes defining your research question, inclusion and exclusion criteria, search strategy, and methods of data extraction and synthesis. Documenting your protocol helps maintain transparency and reproducibility.
- Learn to search effectively: Mastering database search skills is crucial. Get familiar with the databases relevant to your field, learn how to use search operators (AND, OR, NOT), and understand how to adjust your strategy to retrieve the most relevant studies.
- Manage your references: Use reference management software like Zotero, Mendeley, EndNote, etc. These tools can help with organizing bibliographies, removing duplicates, and formatting citations and bibliographies.

Tips related to the Conducting phase:

- Systematic screening: Apply your inclusion and exclusion criteria rigorously. It's often helpful to work in pairs to screen studies, which reduces the risk of bias and ensures reliability.
- Critically appraise your sources: Assess the quality of the studies you plan to include. Tools and checklists, such as the one provided in [60], can guide you in evaluating the validity and reliability of your findings.
- Synthesize data carefully: Whether you're conducting a qualitative or quantitative synthesis, be methodical about extracting data and summarizing findings. Look for patterns, themes, and significant results that respond to the research questions.
- Document everything: Keep detailed records of the search strategy, studies included and excluded, data extraction forms, and assessment of study quality. This transparency is crucial for the credibility of the review.

General tips:

- Seek feedback: Don't hesitate to ask for feedback from peers or supervisors. Their insights can help refine your process and ensure you're on the right track.
- Stay organized and patient: SLRs require meticulous attention to detail and can be time-consuming. Stay organized and patient throughout the process.
- Practice scholarly ethics: Always give proper credit to the original studies and avoid plagiarism by paraphrasing correctly and citing appropriately.
- Keep learning: The field of systematic reviews is always evolving. Stay informed about new methods, tools, and guidelines to refine your skills.
- Attend workshops or seminars: If possible, participate in workshops or seminars on conducting SLRs. These can provide valuable insights and hands-on experience.

Adhering to these tips and committing to learning and applying the SLR process, students will gain valuable skills that will prove beneficial in both academic and professional research.

8 Tools for Performing and Teaching SLRs

The benefits provided by SLRs are offset by the significant investment of time and effort they require. This is due in part to the fact that much of the SLR process involves manual tasks. Therefore, the use of tools can reduce the time and effort involved and can also contribute to increasing the number and quality of published SLRs.

Numerous studies highlight the importance of having tools to support the entire SLR process or specific aspects of it. For example, [38] conducted a feature analysis to compare and evaluate several tools that support the overall SLR process. [2] performed a feature analysis of the current support for desirable requirements in an SLR tool, compiled through several studies within the SE SLR community. Additionally, [39] presented the Systematic Review Toolbox, a Web-based catalogue of tools designed to help reviewers find appropriate tools based on their specific needs. This catalogue is available at systematicreviewtools.comindex.php

Examples of tools that cover almost the entire SLR process include SLuRp [6], SESRA [42], StArt [19], and Parsifal [47]. According to the latest study conducted by [1], the tool StArt has the highest coverage of the desirable requirements from the SE SLR community, although no tool covers all requirements completely.

Despite various efforts to innovate in this area and the availability of specialized tools, spreadsheets—often used in combination with reference managers such as EndNote, Mendeley, and Zotero—remain the most widely used tools for SLRs [1].

To assist in defining the scope of research questions, [14] developed FRAMEndeley, an extension to the Mendeley bibliography manager. The primary aim of this tool is to utilize the articles stored within the Mendeley repository to systematically evaluate the originality of the research topic in question against existing publications.

To conduct a meta-analysis, various tools can be utilized, including Comprehensive Meta-Analysis (https://www.meta-analysis.com), R along with its specific packages for meta-analysis (https://www.r-project.org/), and Stata (https://www.stata.com).

Given the current rise of artificial intelligence (AI), tools such as Rayyan (https://www.rayyan.ai/) and Research Rabbit (https://www.researchrabbit.ai/) have been developed to assist with specific tasks within SLRs. Furthermore, there is significant potential for exploiting ChatGPT and similar tools to support the SLR process [50].

9 Recommended Readings

Among the numerous bibliographic references suitable for systematic literature reviews (SLRs), we particularly recommend the following works:

- *Kitchenham, B. y Charters, S. (2007). Guidelines for performing systematic literature reviews in software engineering, technical report EBSE-2007-01, Keele University*. This technical report, of which there is a first version from 2004, is the reference document used as a guide for conducting SLRs. Its main objective is to detail the guidelines for conducting methodological and rigorous reviews of the literature on empirical evidence in the field of software engineering. It is primarily aimed at software engineering researchers, including doctoral students.
- *Genero M., Piattini M., Cruz-Lemus, J.A. (2023). Métodos de investigación en informática. Amazon*. This book, recommended for Spanish speakers, exhaustively presents both the main primary research methods (surveys, experiments, case studies, action research, and design science) and some SLRs and models of collaboration between universities and industry. Particularly, Chap. 7 provides detailed instructions on how to conduct secondary studies (SLRs and SMSs).
- *Kitchenham, B. A., Budgen, D. and Brereton, P. (2015). Evidence-based software engineering and systematic reviews. CRC Press*. This book presents various research methods to support evidence-based software engineering. In addition to expanding the procedure for conducting SLRs presented in the previous technical report by considering SMSs as well, this book features chapters on primary research methods, such as experiments, surveys, and case studies.
- *Kitchenham, B A., Madeyski, L., Budgen, D. (2023). SEGRESS: Software Engineering Guidelines for REporting Secondary Studies. IEEE Trans. Software Eng. 49(3): 1273–1298*. This paper presents and illustrates SEGRESS (Software Engineering Guidelines for Reporting Secondary Studies), a comprehensive set of guidelines designed for reporting secondary studies in software engineering, based on PRISMA 2020 (https://www.prisma-statement.org/). SEGRESS offers variants that can be applied to both quantitative systematic reviews and mapping studies, as well as qualitative reviews. The authors also provide supplementary material [34] with additional explanations and examples that are critical to understanding how to apply these guidelines.
- *Kuhrmann, M., Méndez Fernández, D. and Daneva, M. (2017). On the pragmatic design of literature studies in software engineering: an experience-based guideline. Empirical Software Engineering 22(6), 2852–2891*. Although there are several guidelines for conducting SLRs and various types of secondary studies, these do not help researchers to deal with the specific difficulties encountered in the practical application of these guidelines. Therefore, this paper presents a pragmatic guide, based on the authors' experience, to assist researchers in designing SLRs, with special emphasis on data collection and selection procedures.
- *Petersen, K., Vakkalanka, S. y Kuzniarz, L. (2015). Guidelines for conducting systematic mapping studies in software engineering: an update. Information and*

Software Technology 64, 1–18. This paper presents an update to the specific guidelines for conducting SMSs as proposed in [51]. Additionally, the paper proposes a guide for the evaluation of SMSs.

10 Conclusions and Future Perspectives

This chapter provides a comprehensive discussion of the teaching of SLRs, emphasizing their importance and usefulness in higher education, particularly for doctoral and master's students. This chapter elucidates the fundamental reasons for incorporating SLRs into academic curricula and delineates the primary benefits these offer within both academic and research settings. The chapter examines a range of effective teaching methodologies and the tools required for conducting SLRs, exploring how they can be integrated into educational practices.

Furthermore, the chapter presents a structured teaching plan for SLRs, which includes clearly defined learning objectives, detailed schedules, and appropriate assessment methods. This ensures that students gain a comprehensive understanding of SLRs and the ability to effectively handle them. Additionally, practical advice is provided for educators and students, which facilitates the enhancement of their skills in conducting these reviews. The significance of open science and the function of reproducibility in SLRs are investigated, thereby reinforcing the value of transparency in research.

The inclusion of a dedicated chapter on teaching SLRs can immensely benefit educators by providing a structured framework to impart essential research skills. SLRs are a fundamental component of academic research, offering a rigorous and methodical approach for synthesizing research evidence. Teaching SLRs to students equips them with critical analytical skills, enhancing their ability to discern patterns, assess the validity of research findings, and understand broader research trends. This skill set is invaluable not only in academic contexts but also in professional settings where evidence-based decision-making is crucial.

Some possible future directions in teaching SLRs are:

- Integrating more technology: Incorporating advanced research software and tools in the curriculum can enhance the efficiency and accuracy of conducting SLRs. Teaching students to use these tools can prepare them for the technological advancements in research methodologies.
- Investigating on the benefits of artificial intelligence (AI) on the SLRs process: AI holds transformative potential for enhancing the efficiency and efficacy of SLRS. By integrating AI technologies, researchers can address several traditional bottlenecks in the SLR process, such as time-consuming literature searches, data extraction, and the synthesis of large volumes of information.
- Interdisciplinary applications: Expanding the teaching of SLRs beyond traditional fields to include interdisciplinary studies can help students apply these skills in varied contexts, from environmental science to public health.

- Collaborative projects: Engaging students in collaborative SLRs can foster teamwork and communication skills, as they work together to compile and analyze complex datasets.
- Continuous professional development: Offering workshops and refresher courses on the latest developments in SLR methodologies can keep both educators and students at the forefront of research practices.
- Assessment and feedback mechanisms: Developing more robust mechanisms for feedback on SLR projects can enhance learning outcomes. This includes peer reviews, presentations, and discussions that encourage critical thinking and iterative improvement.

By continually adapting the teaching of SLRs to include these future directions, educators can ensure that students are not only proficient in conducting systematic reviews but are also innovative in applying this skill across various domains. This will prepare them for the challenges and opportunities of the modern research landscape, fostering a generation of highly skilled researchers who can contribute significantly to their fields.

References

1. Al-Zubidy, A., Carver, J.C.: Identification and prioritization of SLR search tool requirements: an SLR and a survey. Empir. Software Eng. **24**(1), 139–169 (2019). https://doi.org/10.1007/s10664-018-9626-5
2. Al-Zubidy, A., Carver, J.C., Hassler, E.E.: Vision for SLR tooling infrastructure: prioritizing value-added requirements. Inf. Software Technol. **91**, 72–81 (2017). https://doi.org/10.1016/j.infsof.2017.06.007
3. Ampatzoglou, A., Bibi, S., Avgeriou, P., Verbeek, M., Chatzigeorgiou, A.: Identifying, categorizing and mitigating threats to validity in software engineering secondary studies. Inf. Software Technol. **106**, 201–230 (2019). https://doi.org/10.1016/j.infsof.2018.10.006
4. Beecham, S., Baddoo, N., Hall, T., Robinson, H., Sharp, H.: Motivation in Software Engineering: A systematic literature review. Inf. Software Technol. **50**(9–10), 860–878 (2008). https://doi.org/10.1016/j.infsof.2007.09.004
5. Bjørnson, F., Dingsøyr, T.: Knowledge management in software engineering: a systematic review of studied concepts: findings and research methods used. Inf. Software Technol. **50**(11), 1055–1068 (2008). https://doi.org/10.1016/j.infsof.2008.03.006
6. Bowes, D., Hall, T., Beecham, S.: SLuRp: a tool to help large complex systematic literature reviews deliver valid and rigorous results. In: Proceedings of the 2nd International Workshop on Evidential Assessment of Software Technologies (EAST'12), pp. 33–36 (2012). https://doi.org/10.1145/2372233.2372243
7. Brereton, P. Kitchenham, B., Budgen, D., Turner, M., Khalil, M.: Lessons from applying the systematic literature review process within the software engineering domain. J. Syst. Software **80**(1), 571–583 (2007). https://doi.org/10.1016/j.jss.2006.07.009
8. Budgen, D., Brereton, P., Drummond, S., Williams, N.: Reporting systematic reviews: some lessons from a tertiary study. Inf. Software Technol. **95**, 62–74 (2018). https://doi.org/10.1016/j.infsof.2017.10.017
9. Carver, J.C., Hassler, E.E., Hernandes, E., Kraft, N.A.: Identifying Barriers to the Systematic Literature Review Process. In: 2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), pp. 203–212 (2013). https://doi.org/10.1109/ESEM.2013.28

10. Cooper, H., Hedges, L., Valentine, J.: The Handbook of Research Synthesis and Meta-Analysis, 2nd edn. Russell Sage Foundation (2009)
11. Costal, D., Farré, C., Franch, X., Quer, C.: How tertiary studies perform quality assessment of secondary studies in software engineering. In: Proceedings of the 24th Iberoamerican Conference on Software Engineering (CIbSE), pp. 56–69 (2021a). https://arxiv.org/abs/2110.03820
12. Costal, D., Farré, C., Franch, X., Quer, C.: Inclusion and exclusion criteria in software engineering tertiary studies: a systematic mapping and emerging framework. In: Proceedings of the 15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), pp. 1–6 (2021b). https://doi.org/10.1145/3475716.3484190
13. Cruzes, D., Dybå, T.: Research synthesis in software engineering. A tertiary study. Inf. Software Technol. **53**(5), 440–455 (2011a). https://doi.org/10.1016/j.infsof.2011.01.004
14. Díaz, O., Contell, J.P.: Developing research questions in conversation with the literature: operationalization y tool support. Empir. Software Eng. **27**(7), 174 (2022). https://doi.org/10.1007/s10664-022-10204-8
15. Díaz, O., Genero, M., Contell, J.P., Piattini, M.: Adding relevance to rigor: assessing the contributions of SLRs in software engineering through citation context analysis. Comput. Sci. Rev. (in press) (2024)
16. Dixon-Woods, M., Agarwal, S., Jones, D., Young, B., Sutton, A.: Synthesising qualitative and quantitative evidence: a review of possible methods. J. Health Serv. Res. Policy **10**(1), 45–53 (2005). https://doi.org/10.1177/135581960501000110
17. Dybå, T., Dingsøyr, T.: Strength of evidence in systematic reviews in software engineering. In: Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), pp. 178–187 (2008a). https://doi.org/10.1145/1414004.1414034
18. Dybå, T., Dingsøyr, T.: Empirical studies of agile software development: A systematic review. Inf. Software Technol. **50**(9–10), 833–859 (2008b). https://doi.org/10.1016/j.infsof.2008.01.006
19. Fabbri, S., Silva, C., Hernandes, E., Octaviano, F., Di Thommazo, A., Belgamo, A.: Improvements in the StArt tool to better support the systematic review process. In: Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering (EASE'16), pp. 1–5 (2016). https://doi.org/10.1145/2915970.2916013
20. Fernández, A., Insfrán, E., Abrahão, S.: Usability evaluation methods for the web: a systematic mapping study. Inf. Software Technol. **53**(8), 789–817 (2011). https://doi.org/10.1016/j.infsof.2011.02.007
21. Fernández-Sáez, A.M., Genero, M., Chaudron, M.R.V.: Empirical studies concerning the maintenance of UML diagrams and their use in the maintenance of code: A systematic mapping study. Inf. Software Technol. **55**(7), 1119–1142 (2013). https://doi.org/10.1016/j.infsof.2012.12.006
22. Garousi, V., Felderer, M., Mäntylä, M.V.: Guidelines for including grey literature and conducting multivocal literature reviews in software engineering. Inf. Software Technol. **106**, 101–121 (2019). https://doi.org/10.1016/j.infsof.2018.09.006
23. Genero, M., Fernández-Sáez, A.M., Nelson, H.J., Poels, G., Piattini, M.: Research review: A systematic literature review on the quality of UML models. J. Database Manag. **22**(3), 46–70 (2011). https://doi.org/10.4018/jdm.2011070103
24. Genero, M., Piattini, M., Cruz-Lemus, J.A.: Métodos de investigación en informática, 2nd edn. Amazon (2023)
25. Giuffrida, R., Dittrich, Y.: Empirical studies on the use of social software in global software development – A systematic mapping study. Inf. Software Technol. **55**(7), 1143–1164 (2013). https://doi.org/10.1016/j.infsof.2013.01.004
26. Hall, T., Beecham, S., Bowes, D., Gray, D., Counsell, S.: A Systematic Literature Review on Fault Prediction Performance in Software Engineering. IEEE Trans. Software Eng. **38**(6), 1276–1304 (2012). https://doi.org/10.1109/TSE.2011.103

27. Ivarsson, M., Gorschek, T.: Technology transfer decision support in requirements engineering research: a systematic review of REj. Requir. Eng. **14**(3), 155–175 (2009). https://doi.org/10.1007/s00766-009-0080-1
28. Iwazaki, A.Y., dos Santos, V., Felizardo, K.R., de Souza, E.F., Valentim, N.M.C., Nakagawa, E.Y.: Benefits and Challenges of a Graduate Course: An Experience Teaching Systematic Literature Review. In: IEEE Frontiers in Education Conference, pp. 1–9 (2022). https://doi.org/10.1109/FIE56618.2022.9962446
29. Kitchenham, B., Charters, S.: Guidelines for performing systematic literature reviews in software engineering. Informe Técnico EBSE-2007-01. Keele University (2007)
30. Kitchenham, B., Mendes, E., Travassos, G.: Cross versus within-company cost estimation studies: a systematic review. IEEE Trans. Software Eng. **33**(5), 316–329 (2007). https://doi.org/10.1109/TSE.2007.1001
31. Kitchenham, B., Budgen, D., Brereton, P.: Using mapping studies as the basis for further research – A participant-observer case study. Inf. Software Technol. **53**(6), 638–651 (2011). https://doi.org/10.1016/j.infsof.2010.12.011
32. Kitchenham, B., Brereton, P.: A systematic review of systematic review process research in software engineering. Inf. Software Technol. **55**(12), 2049–2075 (2013). https://doi.org/10.1016/j.infsof.2013.07.010
33. Kitchenham, B.A., Budgen, D., Brereton, P.: Evidence-Based Software Engineering and Systematic Reviews. Chapman & Hall/CRC (2015). https://dl.acm.org/doi/10.5555/2994449
34. Kitchenham, B., Madeyski, L., Budgen, D.: Supplementary Material for SEGRESS: Software Engineering Guidelines for REporting Secondary Studies (2022). [Online]. Available: https://madeyski.e-informatyka.pl/download/SEGRESS22supplement.pdf
35. Kitchenham, B., Madeyski, L., Budgen, D.: SEGRESS: Software engineering guidelines for REporting secondary studies. IEEE Trans. Software Eng. **49**(3), 1273–1298 (2023). https://doi.org/10.1109/TSE.2022.3174092
36. Kuhrmann, M., Fernández, D.M., Daneva, M.: On the pragmatic design of literature studies in software engineering: an experience-based guideline. Empir. Software Eng. **22**(6), 2852–2891 (2017). https://doi.org/10.1007/s10664-016-9492-y
37. Liu C., Houdek, R.: Teaching Computer Science Graduate Students. Scholarly Literature Review Techniques. In: Proceedings. Frontiers in Education. 36th Annual Conference, pp. 14–19 (2006). https://doi.org/10.1109/FIE.2006.322349
38. Marshall, C., Brereton, P., Kitchenham, B.: Tools to support systematic reviews in software engineering: a feature analysis. In: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering (EASE'14), pp. 1–10 (2014). https://doi.org/10.1145/2601248.2601270
39. Marshall, C., Brereton, P: Systematic review toolbox: a catalogue of tools to support systematic reviews. In: Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering (EASE'15), pp. 1–6 (2015)
40. Marshall, C., Brereton, P., Kitchenham, B.: Tools to support systematic reviews in software engineering: a cross-domain survey using semi-structured interviews. In: Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering (EASE'15), pp. 1–6 (2015). https://doi.org/10.1145/2745802.2745827
41. Mendes, E., Wohlin, C., Felizardo, K.R., Kalinowski, M.: When to update systematic literature reviews in software engineering. J. Syst. Software **167**, 110607 (2020). https://doi.org/10.1016/j.jss.2020.110607
42. Molléri, J.S., Benitti, F.B.V.: SESRA: a web-based automated tool to support the systematic literature review process. In: Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering (EASE'15), pp. 1–6 (2015). https://doi.org/10.1145/2745802.2745825
43. Mourão, E., Pimentel, J.F., Murta, L., Kalinowski, M., Mendes, E., Wohlin, C.: On the performance of hybrid search strategies for systematic literature reviews in software engineering. Inf. Software Technol. **123**, 106294 (2020). https://doi.org/10.1016/j.infsof.2020.106294

44. Munir, H., Wnuk, K., Runeson, P.: Open innovation in software engineering: a systematic mapping study. Empir. Software Eng. **21**(2), 684–723 (2016). https://doi.org/10.1007/s10664-015-9380-x
45. Noblit, G., Hare, R.: Meta-Ethnography: Synthesizing Qualitative Studies. Sage (1988)
46. Oriol, M., Marco, J., Franch, X.: Quality models for web services: A systematic mapping. Inf. Software Technol. **56**(10), 1167–1182 (2014). https://doi.org/10.1016/j.infsof.2014.03.012
47. Parsifal: Parsifal systematic literature review tool (2015) [Online; accessed 1–July-2024]
48. Paternoster, N., Giardino, C., Unterkalmsteiner, M., Gorschek, T., Abrahamsson, P.: Software development in startup companies: A systematic mapping study. Inf. Software Technol. **56**(10), 1200–1218 (2014). https://doi.org/10.1016/j.infsof.2014.04.014
49. Pedreira, O., García, F., Brisaboa, N., Piattini, M.: Gamification in software engineering – A systematic mapping. Inf. Software Technol. **57**, 157–168 (2015). https://doi.org/10.1016/j.infsof.2014.08.007
50. Petersen, K.: Case study identification with GPT-4 and implications for mapping studies. Inf. Software Technol. **171**, 107452 (2024). https://doi.org/10.1016/j.infsof.2024.107452
51. Petersen, K., Feldt, R., Mujtaba, S., Mattsson, M.: Systematic mapping studies in software engineering. In: Proceedings of the 12th international conference on Evaluation and Assessment in Software Engineering (EASE), pp. 68–77 (2008). https://dl.acm.org/doi/10.5555/2227115.2227123
52. Petersen, K., Vakkalanka, S., Kuzniarz, L.: Guidelines for conducting systematic mapping studies in software engineering: an update. Inf. Software Technol. **64**, 1–18 (2015). https://doi.org/10.1016/j.infsof.2015.03.007
53. Pino, F., García, F., Piattini, M.: Software process improvement in small and medium software enterprises: a systematic review. Software Q. J. **16**(2), 237–261 (2008). https://doi.org/10.1007/s11219-007-9038-z
54. Pizard, S., Acerenza, F., Otegui, X., Moreno, S., Vallespir, D., Kitchenham, B.: Training students in evidence-based software engineering and systematic reviews: a systematic review and empirical study. Empir. Software Eng. **26**(3), 50 (2021). https://doi.org/10.1007/s10664-021-09953-9
55. Riaz, M., Breaux, T., Williams, L.: How have we evaluated software pattern application? A systematic mapping study of research design practices. Inf. Software Technol. **65**, 14–38 (2015). https://doi.org/10.1016/j.infsof.2015.04.002
56. Sierra, J.M., Vizcaíno, A., Genero, M., Piattini, M.: A systematic mapping study about socio-technical congruence. Inf. Software Technol. **94**, 111–129 (2018). https://doi.org/10.1016/j.infsof.2017.10.004
57. Šmite, U., Wohlin, C., Gorschek, T., Feldt, R.: Empirical evidence in global software engineering: a systematic review. Empir. Software Eng. **15**, 91–118 (2010). https://doi.org/10.1007/s10664-009-9123-y
58. Staples, M., Niazi, M.: Systematic review: systematic review of organizational motivations for adopting CMM-based SPI. Inf. Software Technol. **50**(7-8), 605–620 (2008). https://doi.org/10.1016/j.infsof.2007.07.003
59. Turner, M., Kitchenham, B., Brereton, P., Charters, S., Budgen, D.: Does the technology acceptance model predict actual use? A systematic literature review. Inf. Software Technol. **52**(5), 463–479 (2010). https://doi.org/10.1016/j.infsof.2009.11.005
60. Usman, M., Bin Ali, N., Wohlin, C.: A Quality Assessment Instrument for Systematic Literature Reviews in Software Engineering. e Informatica Software Eng. J. **17**(1), 230105 (2023). https://doi.org/10.37190/e-inf
61. Wieringa, R., Maiden, N., Mead, N., Rolland, C.: Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. Requir. Eng. **11**(1), 102–107 (2006). https://doi.org/10.1007/s00766-005-0021-6
62. Wohlin, C.: Guidelines for snowballing in systematic literature studies and a replication in Software Engineering. In: International Conference on Evaluation and Assessment in Software Engineering (EASE), pp. 321–330 (2014). https://doi.org/10.1145/2601248.2601268

63. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering. Springer (2012). https://doi.org/10.1007/978-3-642-29044-2
64. Wohlin, C., Runeson, P., da Mota Silveira Neto, P.A., Engström, E., do Carmo Machado, I., de Almeida, E.S.: On the reliability of mapping studies in software engineering. J. Syst. Software **86**(10), 2594–2610 (2013). https://doi.org/10.1016/j.jss.2013.04.076
65. Wohlin, C., Mendes, E., Felizardo, K.R., Kalinowski, M.: Guidelines for the search strategy to update systematic literature reviews in software engineering. Inf. Software Technol. **127**, 106366 (2020). https://doi.org/10.1016/j.infsof.2020.106366
66. Wohlin, C., Kalinowski, M., Felizardo, K.R., Mendes, E.: Successful combination of database search and snowballing for identification of primary studies in systematic literature studies. Inf. Software Technol. **147**, 106908 (2022). https://doi.org/10.1016/j.infsof.2022.106908
67. Yang, L., Zhang, H., Shen, H., Huang, X., Zhou, X., Rong, G., Shao, D.: Quality assessment in systematic literature reviews: a software engineering perspective. Inf. Software Technol. **130**, 106397 (2021). https://doi.org/10.1016/j.infsof.2020.106397
68. Zhang, H., Ali Babar, M.: Systematic reviews in software engineering: an empirical investigation. Inf. Software Technol. **55**(7), 1341–1354 (2013). https://doi.org/10.1016/j.infsof.2012.09.008

Teaching and Learning Ethnography for Software Engineering Contexts

Yvonne Dittrich, Helen Sharp, and Cleidson de Souza

Abstract Ethnography has become one of the established methods for empirical research on software engineering. Although there is a wide variety of introductory books available, there has been no material targeting software engineering students particularly, until now.

In this chapter, we provide an introduction to teaching and learning ethnography for faculty teaching ethnography to software engineering graduate students and for students themselves of such courses.

The contents of the chapter focus on what we think is the core basic knowledge for newbies to ethnography as a research method. We complement the text with proposals for exercises, tips for teaching and pitfalls that we and our students have experienced.

The chapter is designed to support part of a course on empirical software engineering and provides pointers and literature for further reading.

1 Introduction

Ethnographic studies are an important element of the toolbox for empirical software engineering as they can provide unique insights into software development practice [51].

Ethnography was originally developed as a method to understand foreign cultures by becoming embedded in them and documenting the learning and sense-

Y. Dittrich (✉)
IT University of Copenhagen, Copenhagen, Denmark
e-mail: ydi@itu.dk

H. Sharp
The Open University, Milton Keynes, UK
e-mail: helen.sharp@open.ac.uk

C. de Souza
Universidade Federal do Pará: Belém, Belém, Pará, Brazil
e-mail: cleidson.desouza@acm.org

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2024

D. Mendez et al. (eds.), *Handbook on Teaching Empirical Software Engineering*,
https://doi.org/10.1007/978-3-031-71769-7_21

making that was experienced during the process [26]. Modern ethnography has been developed in an attempt to understand a foreign culture from a members' perspective. With a growing interest in the understanding of subcultures close to the ethnographer, ethnographic research was used to learn more about a wide range of contexts including truck drivers [1], midwives [32], London underground workers [27] and organisational life [58, 64].

In the context of human-computer interaction and computer-supported cooperative work, ethnographic research was used to understand how the members of an organisation made use of technology. Examples for this kind of research include several studies of air traffic controllers in the context of a redesign of the system supporting their perception and control of air traffic [4], of copy machine repair personnel by Julian Orr [40] and of attorneys and paralegals to prepare the design of software supporting their document management [5]. For more than 25 years, ethnography has also been used in the context of software engineering research in order to study cooperative aspects of software engineering. Examples are a study on configuration management [28], a study on the use of social software in distributed software development [23, 24] and an ethnographic study of a small software house to investigate testing practices [39]. As in the original context, the benefit of ethnographic studies of software engineering is the understanding of how and why software teams do what they do to develop software. Ethnography allows understanding and describing collaboration in software engineering from the software engineers' perspective. Ethnographic studies require the researcher to join a development team with the intent to understand the why and how of the team's everyday practices.

The Educators' Corner: Tips for Teaching

- It's important for students to know the origins of ethnography but introduce the approach through a mixture of examples from software engineering and other disciplines so that the relevance to software engineering comes through.
- If you don't have (much) experience of ethnography yourself, then identify colleagues with experience to engage in your classes.
- It's hard to see another person's point of view, offer your students guidance on emotional intelligence and how to see different perspectives.
- Present examples of existing ethnographies in software engineering for the students to facilitate their writing.
- Take an iterative approach to topics, e.g. provide some background, set an exercise, discuss the exercise and ask students to repeat the exercise but with a deeper understanding.
- Embrace the use of ethnography with other techniques, e.g. interviews, as these are often required in the field.
- Because activity in the field is fluid, ethical dimensions of the study need careful attention.

When we use the term ‘practices’ in this chapter, we talk about the established way a software team goes about developing and evolving software. Social practices, like cooperative software engineering, are based on an understanding of relevant objects, representations and tasks, on implicit and explicit rules and on a common goal (for further readings, see Sect. 7).

Understanding software practices using an ethnographic approach aims to understand how and why the software development practices make sense from members’ point of view, putting aside ideas on how to improve the observed practices for the time being. This can be hard for a software engineering researcher. As engineers and software engineering researchers, we are trained to design and create things and to improve the methods used to this end. We tend to focus on problems and shortcomings, rather than aiming to understand how a complex activity such as software development actually takes place or what enables a team to produce software at all. Ethnographic research invites us to temporarily put aside or ‘bracket’ [22] our assumptions of how software development should take place and try to understand how developers make things work no matter what.

In return, the results of ethnographic research help us understand the rationale behind even seemingly disadvantageous behaviour, which in turn provides us with important information when designing tools and methods. An example can be found in [61, 62]. The articles present an interview study triangulating an ethnographic study of software architecture practices for a software product. The researchers were surprised at the lack of any software architecture documentation and the unwillingness of the team to produce and maintain such documentation. A triangulating interview study revealed that this reluctance towards software architecture was shared by architects and tech leads of other products. The only exception, an open-source project, also provided us with a rationale for this attitude: the existence of architecture documentation can prohibit discussion between the architects and the developers with the effect that the architects might not know how the developers change the architecture or where the architecture creates problems for the development and might need to be revised. This, in turn, means that tools for designing and communicating software architecture should be designed to support architect-developer discussions and not replace them. Had we not taken the reservations of the developers and architects seriously, we would not have understood that there was a ‘good reason for bad architecture documentation’ [62].

Ethnographic studies can also lay the groundwork for practitioner support tools that are accepted by the community because they are based on an understanding of practitioners’ point of view. An example of this is the motivating jenny project which ran a multi-sited ethnographic study to understand developers’ security behaviour. This project resulted in a number of insights including a model of security behaviour [35] that underpins a set of four practitioner packs designed to improve the security culture in a team or organisation (downloadable from motivatingjenny.org). These have been downloaded thousands of times and have been used in education and in industrial settings.

Our previous, more theoretical and programmatic article in *Transactions on Software Engineering* [51] provides examples from empirical software engineering

where ethnography has been used. This chapter complements that article with support on how to implement and teach ethnographic research in a Software Engineering context. The goal of this chapter is twofold. On the one hand, we introduce ethnography as a research method for software engineering with a hands-on approach. We aim to provide concrete how-to support for researchers starting to use ethnography. Our intended reader in connection to this goal is a researcher, e.g. a PhD student, who wants to explore ethnography as a possible research method. On the other hand, we provide guidance for teachers of a postgraduate course on empirical research in software engineering. The chapter comprises what we see as the minimum contents of a module on ethnography as an empirical method for software engineering.

In our tutorials, people have often asked us what the difference between ethnography and other qualitative methods is, such as contextual inquiry and grounded theory, and why not just do interviews? A simple response to these questions might include that contextual inquiry is based on an apprenticeship model, grounded theory seeks concepts in the domain that can be developed into a theory and in interviews participants' accounts will necessarily be partial and rationalised. Ethnographic research, on the other hand, focuses on a holistic view of the practice being studied, has research questions that evolve over the course of the study and aims to capture activity as it happens together with practitioners' rationale. But there is more to it than this. This chapter will provide a deeper answer to these questions.

The remainder of the chapter is structured as follows: Sect. 2 discusses whether ethnography is the right choice from three perspectives: the role of ethnographic studies in relation to your specific research context, the type of research question it can address and what is required of the researcher. Section 3 discusses the planning of an ethnographic study. Section 4 covers important topics such as field work and addresses issues ranging from the role the researchers take on in relation to the team, to how to design and keep a field diary. Section 5 introduces the analysis of ethnographic data, and Sect. 6 covers research ethics for ethnography. Section 7 concludes the chapter and provides suggestions for further reading.

? Exercise 1: Identify an Ethnographic Research Project

The exercises in this chapter together result in the design of an ethnographic study project. If you already plan to do ethnographic research, feel free to use your project throughout the exercises.

If you use other methods in your research, try to find an angle to your research that could benefit from a real-world investigation. For example, if you work with mining software repositories (MSR), you might consider conducting an ethnographic study of one of the projects that would shed light on the data you base your MSR study on.

Formulate a short text (about half a page) motivating your study and a research question related to understanding relevant development practices from a members' point of view.

The Educators' Corner: Exercises

Throughout the chapter, we propose a number of hands-on exercises that encourage students to apply the contents in a concrete situation. Whereas the introduction to ethnography targets both students and more senior researchers, the exercises are directed to a student designing their empirical research project. The order of the sections in this chapter is designed to support a logical progression in these exercises.

The exercise for Sect. 2 asks students to formulate an adequate research question for their own or a fictional ethnographic study. Section 3 discusses the planning of the ethnographic study, accompanied by an exercise to develop a preliminary plan for their own or a fictional study. Section 4 describes the field work. Here we propose an exercise that sends the students out to observe an everyday situation to gain hands-on experience. In a second exercise, the students are asked to prepare the concrete field work. Section 5 discusses the analysis of the field material. An experienced researcher might bring their own empirical data to the course and provide the students with hands-on experience with field material. As a fallback exercise, we propose the development of an analysis strategy for their own or a fictional research project. The discussion of ethnographic research ethics in Sect. 6 should enable the students to review their study designs from an ethics perspective and discuss the ethical challenges of their research design. Section 7 does not have specific exercises.

2 Is an Ethnographic Study the Right Choice?

Here we consider the question of whether or not an ethnographic study is the right choice from three different perspectives:

1. The context of your research
2. The kind of research question you want to answer
3. What it means for you as the researcher because in an ethnographic study, the researcher is the research instrument

This section will discuss these three perspectives to help you decide whether to use ethnography as a research method.

2.1 The Context of Your Research

This perspective considers where the intended study fits in the wider context: what role might an ethnographic study play in the research? In social sciences, where

ethnography originated, studies focus on understanding phenomena. They are used to understand better how (sub) cultures operate or how groups of people organise their social life. In HCI, ethnographic studies are used to inform the design of technology from a user's point of view so that the designers might better appreciate who they are designing for. In software engineering, ethnographic studies may also be used to understand or to inform design, but research often goes beyond understanding towards improving methods, techniques and tools. Understanding current practice is a good starting point to inform changes of any kind, not so that current practice can be replicated in the new order but to appreciate why things are done the way they are so that there are no unintended consequences when things change. A quite famous example is the long time it has taken to replace the paper flight strips used by air traffic controllers with digital versions. In 1999, Wendy MacKay commented that 'Our observations have convinced us that we do not know enough to simply get rid of paper strips, nor can we easily replace the physical interaction between controllers and paper strips' [38]. She wrote this after being inspired by her work and previous studies about flight strips, which started in 1992. A blog written in 2017 on the subject demonstrates how cautiously this change was eventually made [18].

In our analysis of ethnographic research in empirical software engineering [51], we identified four roles for ethnographic research, all of which start by considering 'how things work in practice'.

2.1.1 To Strengthen Investigations into the Social and Human Aspects of Software Engineering

Since ethnography's origins are in understanding cultures and communities, it is not a surprise that ethnography has been used in this way. It is inevitable that some insights regarding social and human aspects will emerge whatever research question is pursued since the approach champions the members' point of view.

For example, Lopez et al. [35] were interested in investigating how non-specialist software developers engage with security in practice. This explicitly focuses on the human aspect of software developers. They used a multi-sited study and a mixture of research methods. Their results included a set of episodes where non-specialists engaged in security activity and five typical behaviours that characterise how individual developers respond to security concerns to meet the demands of particular circumstances. A key characteristic of this work was that it emphasised security from the developers' perspective. The results form a framework that managers and teams can use to recognise, understand and alter security activity in their environments (motivatingjenny.org).

2.1.2 To Inform the Design of Software Engineering Tools

For a new tool to be successful, it must support the task being performed, e.g. bug localisation. But for it to be useful to practitioners, it must also fit into their workflow and support how they and their organisation perform the task. Focusing on a task in isolation gives only a partial view, and however clever it may be, if it's not usable, then it won't be used. So understanding the context of tool use is as important as understanding its technical and functional requirements.

But there are other considerations too. As with much software development, it can be easy to jump straight into designing a system that seems to address a user's problem [48]. However, if the goal is to develop new software engineering tools, then don't start the design until you are sure that the proposal will be useful. This can be achieved by sketching a number of potential tools based on observations and then collecting evidence to support or refute their value, e.g. by working with practitioners and analysing your data overall to evaluate the ideas. For example, de Souza et al. [9] were interested in the role of application programming interfaces (APIs) in the coordination of software development work in large-scale organisations. An initial observation led them to think that integrating the versioning software with email might facilitate the team's work, but further data collection made it clear that this was only part of the situation, and focusing on a tool to support the team's overall communication would bring them more benefit. So instead of developing something to integrate two existing tools, they developed a tool that supported the identification of dependencies between developers, which allowed developers to coordinate their efforts more effectively and reduce unnecessary work.

2.1.3 To Improve Development Processes

Ethnography can be a starting point to discuss and implement improvements to work practices, together with the practitioners observed. Ethnographic research then becomes part of action research. Two chapters in this book by Staron and by Dittrich, Bolmsten and Seilein introduce action research. A critical point is that the decision about the intervention and the evaluation of its implementation should keep the members' point of view, both for ethical and methodological reasons. Dittrich et al. developed Cooperative Method Development (CMD), an action research approach combining ethnographical and ethnomethodological inspired empirical research with the improvement of software engineering tools, methods and processes [13, 17]. CMD is explicitly designed to keep both the work practice focus and the developers' point of view of the initial research throughout the action research cycle(s).

Unphon's thesis provides an example of action research [61]. The research cooperation with a company developing software products for simulating hydraulic systems focused on the evolvability of these software products. The researchers collaborated with the team responsible for the product simulating open one-dimensional water systems like rivers and creeks. The action research introduced

lightweight software architecture techniques for high-level design, developed a lightweight Architecture Level Evolvability Assessment method for focussed discussions of design decisions with relevant stakeholders and introduced lightweight architecture compliance techniques using the built system. Introductory research with the team in order to understand development practices as well as investigate the structure of the existing software product resulted in a framework for understanding the influence of the organisation of software development and business on the architecture of the software [62]. The research results emphasise the need to adapt software architecture methods and tools to support the continuous evolution of software products: architecture design and evolution take place as part of everyday evolution; architectural practices need to support the software architect to keep up with the changes of the software and the emerging requirements that might challenge the architecture and evolvability should be a quality to be considered in regular software architecture design discussions.

2.1.4 To Inform Research Programmes That Do Not Have Ethnography at Their Core

It is common for research outputs such as tools and new processes to be ignored by practitioners, in favour of innovations suggested by other practitioners. Understanding the context of software engineering takes the researcher one step nearer to suggesting an innovation that is acceptable to practice. In particular, an ethnographic study may help a research programme by articulating more specific, relevant research questions.

For example, our programme on agile software development started by looking at the realities of XP in practice, i.e. how was the approach implemented [53]? Ethnographic studies in this context led to other studies and research questions that focused on the role of physical artefacts [55], information flow between stakeholders [65] and collaboration in dispersed teams [11].

But ethnography can also be used to inform research programmes by providing context grounded in practice for an existing research focus. For example, Capiluppi et al. [6] investigated the evolution of code within an agile development project. This led to a number of observations regarding how the agile code base evolved. Using an ethnographic study as context allowed the researchers to:

- Characterise the development process and practices clearly.
- Provide alternative interpretations of phenomena found in the data, e.g. where the size of the code base changed significantly, they could trace an event through the ethnographic study to explain that a new library was loaded.

Ethnographic observation in this context has the benefit that it is not dependent on the report of practitioners, as it is in interviews. When collecting reports from practitioners, their responses will inevitably be partial and informed by current events. For example, they might not consider certain aspects of everyday practice to be relevant to the questions asked, or they might not report practices that they

consider to be informal or not accepted. However, these aspects may be exactly the characteristics that render the specific way of developing software possible and hence would be important from a research point of view. The design of studies complementing and providing context to other research may not be driven by an independent research question, but the research question might be decided based on the overall research interest.

? Exercise 2: What Role Do You See for Your Ethnographic Study?

Consider the four roles above, and assess your own context against each one. To which does it fit most closely? It's ok to align with more than one at this stage. The detailed design will help you narrow it down.

Describe the role that you see for your ethnographic study in a short paragraph, and add it to the study design started in Exercise 1.

! The Educator's Corner: Pitfalls

- When conducting an ethnographic study with the aim of investigating social and human aspects of software engineering, it is easy to become overwhelmed by the huge number of focus points, SO track the possible foci and address them one at a time.
- When conducting an ethnographic study with the aim of producing a new tool, it can be tempting to start designing before a full picture has been obtained, SO start by hypothesising what kind of tools would be useful, and collect data to validate those hypotheses.
- When conducting an ethnographic study with the aim of producing a new tool, it can be tempting to focus on specific instances rather than the broader picture, SO focus on the analytical results rather than the data when hypothesising about potential tools.
- When conducting an ethnographic study with the aim of improving development processes, it can be easy to suggest changes based on textbook versions of processes, SO make sure to engage practitioners in any suggested modifications.
- When conducting an ethnographic study with the aim of informing other programmes of research, confirmation bias can easily cloud your observations, SO be sure to seek confirming and disconfirming evidence.

2.2 The Kind of Research Questions You Want to Answer

This subsection considers the kind of research questions you want to answer. The specific detail of the question may change as the study progresses (see Sect. 4), but whether an ethnographic study is the right choice depends on the kind of question you want to answer.

The first thing to note is that ethnographic research, like other qualitative research, is driven by research questions rather than hypotheses derived from theory. Research questions ask 'How' and 'Why' and 'What are the characteristics of' questions rather than 'Is X better than Y' or 'Will this technique make programmers more productive?' kind of questions. For example, 'How do software practitioners develop systems using XP?' rather than 'Is single programmer coding more productive than pair programming?', and 'Why don't developers adhere to the company's security policies?' rather than 'Does structuring the manual in this way help developers produce more secure code?' and 'What are the characteristics of a technology adoption?' rather than 'How did the ideas of Simula develop into Java?'

Other techniques and methods introduced in this book will support answering other types of question.

The strength of an ethnographic approach is that it emphasises the point of view of the participants, i.e. the members of the community under study, who are often called informants or interlocutors in ethnographic work. It therefore allows the researcher to understand better why things are the way they are. It brings research closer to practice and hence can make the results more acceptable within practitioner communities. For example, it may seem strange to a researcher to see agile practitioners using both a physical storyboard and a virtual one, especially because this entails duplicate work in keeping them both up to date. However, this is common practice where co-located or hybrid teams are deployed because of the different and complementary collaboration and communication benefits it provides [12].

? Exercise 3: What Type of Research Question Do You Want to Answer?

Review your research question(s) from Exercise 1. Are they 'how', 'why' and 'what characteristics' questions, as discussed above? If not, can they be recast appropriately?

Revise your study design if needed, and make sure to adjust the whole text to any new formulation of your research question(s).

2.3 What Ethnographic Studies Require from the Researcher

To help decide whether or not an ethnographic study is the right choice, this section considers the four main features of ethnographic work: the member's point of view, the ordinary detail of life as it happens, the analytic stance and thick descriptions. For each of these, we present considerations on whether the researcher (you) are prepared and willing to apply the approach appropriately, e.g. to write thick descriptions, listen to the informants' point of view and modify your research focus. The information in this section will help to assess this and will set your expectations for what it means to implement an ethnographic study.

2.3.1 The Members' Point of View

We would argue that it is always important to understand the point of view of practitioners whatever is the focus of the research, for example, whether developing a new tool, looking to improve the development process or simply trying to understand better how something works in practice. As discussed in Sect. 2.1, taking account of the members' point of view doesn't restrict innovation, but instead understanding the rationale for the current way of working before introducing changes helps avoid unintended consequences. Understanding the member's point of view is all about understanding what is important for the informants so that this can be taken into account.

In other disciplines, ethnographic studies involve the ethnographer studying a culture that is unfamiliar to them, e.g. in terms of language, customs, cultural norms, etc. These studies may need to last for months or years. As a software engineer conducting an ethnographic study in a software development environment, this will be less of a challenge to overcome. Understanding someone else's perspective is still hard, but much of the basic context of software development, e.g. IDEs, development processes, programming languages, etc., will be familiar. One of the benefits of this is that ethnographic studies don't necessarily need to take months or years. Focusing on the members' point of view will entail getting used to the organisation's environment and the specific details of the team's work, but the learning curve is less steep than it would be for someone unfamiliar with software development.

The familiarity of the researcher with the tools, techniques and methods of the informants can make it difficult to keep the member's point of view in mind when observing how software development takes place. It is very easy to apply the knowledge from your own education and research and slide into a 'this can be done better' attitude. Such an attitude might even be provoked by the practitioners asking the researcher, who supposedly is an expert, whether they have any recommendations. Here it is important to separate observations from analysis for the duration of the field work and put your own judgement aside.

2.3.2 The Ordinary Detail of Everyday Life

The ordinary detail of everyday life is important because it exposes how tasks are addressed, the issues that are relevant or not and how informants approach tasks. For example, a series of studies with agile software development teams exposed that the colour of cards placed on a physical scrum board and the way in which they are handled carry meaning beyond what was written on them [54]. This raised questions of whether digital storyboards would support the same kind of processing and collaboration as physical boards.

This also means that the research will be conducted in the field or 'in the wild' rather than in a controlled environment and the research will aim to not disturb or control normal behaviour.

Equally, informants' language is relevant. For example, if the informants' native language is different to the ethnographer, what might the impact be? If the domain of study is highly technical, then the ethnographer would do well to understand the domain too. Performing an ethnographic study in an unfamiliar technical domain will have an impact on the timescale (as mentioned above) but also may lead to misinterpretation of activities and tasks. The gender of the researcher might both influence the field situation and the interpretation of the observation.

Given that ethnography emphasises what is important from the members' point of view and that the researchers won't know what they might find before entering the field, there is a strong chance that the research focus will evolve. A key issue to consider is whether the research context and design is suitably flexible to allow for the research focus to change. For example, if your focus is developing a new tool to identify areas of technical debt but your informants explain that their concerns are driven by improving the security of the code, how would that affect the research?

Although the detail of everyday life is observed, it is not the detail itself that is important but the significance of the detail (see Sect. 2.3.3).

2.3.3 The Analytic Stance

An ethnographic account is not just a description of what is seen but is crafted out of an analysis and interpretation of what was found in the field, explicating why things are the way they are. The analytic stance is related to the members' perspective. The idea is to understand and uncover the rationalities of practices and how the practitioners' behaviour makes sense, even though it does not match the researcher's own understanding of how software development should take place. This can be problematic to those unused to the approach.

One example to illustrate this analytical stance can be found in [49], where a recording of a steering group meeting addressing a major change in a method and tool development project is analysed. The analysis shows in detail how the steering group relates the actions to take the development further to both the new plan for the project and the company-wide project model. The analysis shows that the steering group takes a decision to deviate from the project model in order to assure that the

work in the project can progress using, nonetheless, the very same company-wide project model to make the deviation visible and accountable in the organisation. The analysis thus sheds light on the rationale behind deviations from plans and company-wide project models, and at the same time, it shows the importance of a company-wide project model to communicate both behaviour according to it and deviations from it to other actors in the organisation.

Not every analysis needs to be as fine grained as the interaction analysis performed in that article. However, the analytical stance requires the researcher to be able to relate the observations here and now to the spatial and temporal context and to the goal and purpose of the observed activities.

2.3.4 Thick Descriptions

An ethnographic study results in a comprehensive and detailed set of data and a detailed account of the findings. The account aims to communicate a broad picture of the study environment and activity. It needs to show how the researcher has arrived at their conclusions or recommendations. The set of data is specific to that site and hence is not suitable for statistical generalisation although it may be suitable for analytical generalisation. If you decide to undertake an ethnographic study, this is an important point to remember. In an area where generalisation and statistical significance of results are expected, having a core ethnographic study may be challenging to convince others. The thick description can be used to defend the study and its results, but it cannot justify a statistical generalisation. An ethnographic researcher needs to acquire a different perspective on justification. This may be particularly pertinent for publication and if you are a PhD candidate.

? Exercise 4: Are You Ready for This Kind of Research Study?

Is there anything in this section about what it means to do an ethnographic study that surprised you or that seems too difficult? Consider each of these features, and write a list of advantages and a list of disadvantages from your point of view. Review these lists, and refer to them as you work through the rest of this chapter and the design of your own study.

3 Planning an Ethnographic Study

Ethnography belongs to the category of flexible research designs [47], where both the research focus and the concrete field work are expected to be adapted during the research process. However, this does not imply that planning is redundant. An ethnographic study still needs to be planned. It is important that both the researcher and the practitioners have a shared understanding about what might happen once the

researcher joins the project. In addition, the plans should be regularly revisited and adapted and everybody kept aligned throughout the research.

3.1 Finding a Site for Field Work

One of the first things is to identify a suitable project to collaborate with. Often, the ethnographic study will have been planned and the focus decided upon through prior contact between the researcher, the company and relevant development team members. In other cases, the ethnographic researcher might identify a research interest and research question first, and then find a company or open-source project that fits this focus.

When working with a company, the ethnographic study might affect their work, so the company and the project team need to agree to the study and the plan. As ethnography is an extremely flexible way of doing research, good communication with the project and, in many cases, the management of the company is needed to allow for coordination and replanning. One way of doing this is to create a 'steering committee' for the research collaboration, consisting of the researcher, the PI of the project, a project contact and the person responsible for the research collaboration in the company. This committee would meet regularly to discuss whether changes to the logistics of the field work are needed and to make sure that resources necessary for the ethnographic study (access to documents, code and the like, time for interviews, etc.) are provided. Exactly how this group works will depend on the field site circumstances, but regular alignment across all parties will keep the study on track.

In other cases, the researcher could decide to become part of an open-source community. This in turn will require deciding on the role in the community the researcher would want to take. Are you able to and interested in participating in the open-source development, i.e. coding? Could you contribute in other ways to the community? Open-source members, especially if they take on core roles like maintainers, are often as busy as members of corporate software engineering teams. Either your contribution to the community during the field work or the research results could be received as a return for the time the community members spend with you.

Planning the ethnographic study needs to take research ethics into account as well. Due to the complexity of this topic, research ethics are discussed on their own in Sect. 6.

The following discusses five dimensions to consider when planning an ethnographic study, especially in software engineering.

3.2 Participant or Non-participant Observation

The best way to understand the culture and practices of a community under study is to become part of that community. Studying software engineering practices as a software engineer often allows for participation. For example, researchers can become members of open-source communities by participating in the development. However, commercial companies may not be so happy for researchers to be modifying their code, and highly specialised software—for example, software for modelling hydraulic systems [61]—requires an expertise that software engineering researchers would normally not master. In such cases, the researcher needs to find a role that allows them to observe and become a culturally competent member of the community. One way could be to act as a newly employed project member, an apprentice in the specific development context. In the above case of hydraulic simulation software that required expertise in differential equations on a PhD level, the field worker took on documenting the architecture of the re-engineered software to support its usage and customisation. Another example was in an early ethnographic study where the researcher took on an administrative role [37]. Both of these examples resulted in the researcher getting embedded in the work of the team in a way that was acceptable to all concerned. Deciding the role of the researcher in the field needs to be taken together with the collaborating project team. Also, other stakeholders like management and, in some cases, even clients of the organisation need to be consulted.

It will not always be possible to participate in the development, and a more passive observation may be the only option. In this case, the researcher will sit quietly and observe, listen and take notes, but then it is important to keep alert for particularly pertinent activity.

3.3 Duration of Field Work

How much time does the researcher need to spend with the software team? In our presentation of published ethnographic studies [51], there was a wide variety of field work durations. The traditional answer of ethnographers would be until the researcher can act as a competent member of the culture or until the researcher stops learning new things. As in other qualitative research methods, saturation of the observations is an indication for ending an ethnographic study. Here, the overlap of expertise when researching software engineering practices as a software engineering researcher might shorten the time necessary: although the specific software practices might be new, the field worker shares a common ground with the observed teams in the form of shared professional expertise. However, saturation might not be possible, e.g. if a software development project ends before saturation is reached. On the other hand, iterative processes such as agile software development might allow repeated observation of all relevant practices in a comparatively short time.

In many situations, it is not necessary to spend all the time with the observed projects. For example, the field worker might choose to spend 3 or 4 days each week with the project, which enables the researcher to start analysing the field material in parallel with the field work. That way the field work duration can be adjusted to fit with time periods that are meaningful from the point of view of the researched practices.

3.4 Space and Location

Ethnographic field work should take place where the culture and practices that are studied take place. But where does software development take place? Software development is both anchored in the physical world, where software developers are located, and in the digital realm, where the software is located. Observations often need to combine physical observation and an understanding of how the physical actions relate to the development environment, digital documents, changes to and management of the source code as well as the effects on the deployed software.

Especially when cooperation and collaboration in distributed development is the focus of an ethnographic study, participatory observation on only one side risks neglecting the remote conditions for collaboration. To address this potential bias, multi-sited ethnography emphasises the need to do ethnographic field work on all the locales involved. Multi-sited ethnography entails the researcher entering and being accepted at different sites which might require longer stays, e.g. one or more weeks, at all places where relevant project members or stakeholders are situated.

As an additional challenge, especially after the lockdowns due to the pandemic in 2020 and 2021, many companies moved from co-located or distributed development to hybrid work organisations, where developers partly work from home, leading not only to distributed but dispersed development, where every member of the team is in a different location [52]. Coordination and cooperation in these circumstances more and more resemble open-source development. Ethnographic research here has to follow the way software development is organised.

One of the core challenges with researching at least partially virtual activities is that it might be hard to follow an activity in a complex distributed development environment that both has physically visible parts, e.g. workstation layout and virtual elements, e.g. changes to code and deployed software. For example, Begum [3] describes the handling of a bug report: the bug report comes into existence through registration of the bug by a support engineer, followed by reproduction of the erroneous behaviour and a root cause analysis by a software engineer who prepares the discussion of the defect in a triage meeting, which in turn results in a strategy to fix the defect. The correction of the defect might again require collaboration between different teams responsible for different parts of the product. To understand what is discussed at the different meetings and to understand the rationale behind decisions might require at least a superficial understanding of the organisation of the source code.

Ethnographic field work here can follow an artefact, e.g. a bug report or the software architecture documentation, or it might focus on the activities of individuals, e.g. the software architect of (a part of) the software or a group of developers. Different forms of 'virtual ethnography' have been developed as a way to observe Internet-based cultures (e.g. [29, 33, 43]).

Regardless of how well the field work is planned and prepared, even in smaller projects, the ethnographer will not be able to observe all relevant activities that are going on. Observations might need to be complemented with other data collection methods, like informal, in situ interviews, semi-structured more formal interviews or group interviews, e.g. in the form of a project retrospective [15].

3.5 Theoretical Underpinning

Ethnography often is seen as a bottom-up research method where the development of themes and concepts is based on reflective and inductive analysis. Theories are then developed by comparing results of several ethnographic studies. In order to relate ethnographic studies with each other, the themes and concepts developed in earlier studies have to be connected to the field work and analysis of the later studies, and this can be done via theory. One example is the social theory of learning by Lave and Wenger [34]: comparing findings from widely varying studies led to the formulation of a set of related concepts, a theory. This theory can then be used either as a focus for the field work in new ethnographic studies exploring social learning in different contexts or to discuss the analysis of field material when peer learning emerges in the reflective and inductive analysis.

Theories can be seen as tools to think with rather than a claim for quantifiable correlations that are or can be supported by experiments. For example, the social theory of learning talks about 'communities of practice' [34] as an important facilitator to develop and maintain expertise in professional environments. 'Communities of practice' as a phenomenon can be identified both through the way members refer to each other and how knowledge is shared with new colleagues and among established members. Using the concept, for example, to understand the onboarding of novices in a software engineering team, can help structure and focus the observations in the field.

A danger of using theory from the outset, though, is that the focus that a theory provides can become a bias in the empirical work. It can prevent the researcher from paying attention to aspects of the observed practices that do not fit the theory. To prevent theory-based biases, the same techniques used for other biases can be applied (see Sect. 4.2).

Ethnography research sees software development teams as sub-cultures or as social practices. Social theories that focus on social practices and how they reflexively constitute social structure therefore might fit well with the observations and findings and help to develop relevant insights. There are several such practice theories. For example, activity theory [19] focuses on how human goal-directed

activity is mediated by tools and also by rules, distribution of labour and community. Activity theory has, for example, been used to better understand tool support for distributed development [60]. Dittrich et al. [16] and Dittrich [14], use Schatzki's [50] concept of social practices and Knorr Cetina's [7] concept of epistemic practices as inspiration to understand the evolution and design of software development practices. Here, the way members of a software team adopt and adapt methods is at the core of the research.

Distributed cognition [31] emphasises the distributed nature of cognitive systems and focuses on collaborative work and the use of artefacts and representations. It results in an event-driven description which emphasises information and its propagation through the cognitive system under study. This has been used, for example, to understand how the structure of story cards and agile boards support collaboration [54] and how UX information is handled by agile development teams [65].

The decision to consider theoretical frameworks can be taken based on the research intent, for example, if the intent is to design and improve tools, the use of a theoretical framework that provides a vocabulary to identify and discuss the role of tools can be an advantage. As mentioned above, theory can also be used to interpret the data afterwards, e.g. Sharp et al. [56] used cognitive dimensions to analyse representations that were collected using distributed cognition as a framework.

Giuffrida and Dittrich's article [25] provides an example where ethnographic research is used to develop theory. They further develop concepts from coordination and communication theory to explain widely different success rates of student projects in distributed student projects. The initial analysis showed that successful projects used the instant messaging channel Skype provided to not only coordinate but also agree on coordination. In order to explain both what was going on and the differences, Genre theory and concepts based on articulation and coordination theory were combined and further developed.

The Educators' Corner: Tips for Teaching

- Spend time discussing what data to collect because observational data, e.g. meeting notes, is more subjective than other kinds of qualitative empirical data, e.g. transcripts of interviews or meetings.
- When discussing what data to collect, show the students that different levels of detail are possible and that they need to decide which level is relevant for them according to the research interest they have.
- When discussing what data to collect, show the students that the level of detail changes as the research progresses.
- The relationship with the gatekeeper and other informants needs ongoing management.

? Exercise 5: Planning Your Ethnographic Study

Expand your study design developed so far with a concrete plan. If you know the organisation and team you plan to collaborate with, describe the company, OSS community or other organisational anchor, the team and the project. If you are working with a fictional ethnographic study design, describe the ideal organisation or OSS community, a team that would be typical for your research question and an example of the kind of project you would like to engage with. Then describe and motivate:

- Whether you anticipate doing participatory or non-participatory observation.
- How long (elapsed time) and for how much time (percentage of a day or week) you anticipate the field work to go on.
- Where and how you expect the observation to take place.

4 Implementing Your Ethnographic Study

Planning can only prepare you so far for the research, but here is where the 'rubber hits the road'. As Fetterman [20] puts it: 'The most important element of field work is being there - to observe, to ask seemingly stupid questions, and to write down what is seen and heard'.

4.1 Gaining Access and Starting Up

After reading the previous sections, we expect the reader to have a clear understanding of what ethnography is, what kind of research questions it is best suited for and the issues to consider when planning an ethnographic study. Furthermore, a 'field site' to be studied has to be decided, i.e. a software development team, an open-source community or organisation. However, as one can imagine, starting an ethnographic study is not only about 'showing up' to observe what is going on at the field site.

In general, it is important to have a key informant (also called a 'gatekeeper') that can facilitate the researcher's access to the informants. This can be a project manager, team lead or a scrum master. This key informant is a person who is trusted, and possibly respected, by the informants and is willing to introduce the researcher to them so that they can start the data collection. In summary, this key informant negotiates access with the informants before the researcher goes to the field. Often, a good idea is to let this key informant introduce the researcher and suggest one or two initial members that are willing to be observed or interviewed. By interacting

with these initial members, the researcher will have a chance to meet other team members and can negotiate with them additional opportunities for data collection.

The key informant is also the person that the researcher will look for to ask clarification questions regarding the project being studied, its context, the diverse roles being played by the informants, the software development organisation and the software being developed. This information often will also indicate documents and tools that the researcher should be aware of, and if necessary, the key informant will make the necessary arrangements so that the researcher can access these documents and/or tools. The key informant can be thought of as a facilitator who will allow the researcher to conduct the ethnographic study.

Part of starting up the actual field work is also to agree on the legal side of the collaboration and develop and sign cooperation agreements containing non-disclosure agreements and agreements about the intellectual property that might be developed as part of the research project. As part of starting up the field work, the researcher also needs to consider the research ethics relevant for this project and decide, maybe together with the key informant, on how to handle the informed consent. The latter topics are discussed further in section 7.

? Exercise 6: Observation

Select a social context, which you can observe without disturbing the situation. This could be, e.g. how people behave in the canteen when queuing for food and paying, or it could be how people behave on public transport.

Take notes on how people behave in this context. You might take different foci, e.g. how a specific person gets through the system, the role of artefacts like time tables or menus or events that change/disturb activity.

Formulate a text describing your observation to a traveller from outer space.

The Educators' Corner: Tips for Teaching

- Don't impose your point of view. Let your student express their perspectives.
- Challenge your student's observations but gently.
- Spend some time focusing on the use of language. How informants talk about their work or aspects of their organisation is very informative, but attention to this kind of detail is not often emphasised in software engineering classes.

4.2 Handling Your Preconceptions

One of the challenges of ethnographic research for a software engineer studying software engineering practices is that the researcher brings an education to the field that states how software should be developed. For example, having attended a software engineering course, the field worker might expect a 'stand-up' meeting to take place in situ by people actually standing up. However, in virtual software development, a stand-up meeting might mean that people sit in front of the screen communicating using a virtual meeting environment [16]. Similarly, the textbook version of pair programming describes a 'driver' and 'navigator' role, but in practice, pairing rarely adopts these roles formally, as the intensity of development results in a much more fluid exchange of activities [45]. So how can a researcher handle such a situation where the observed team is not doing things 'by the book'?

Ethnographic research talks about 'bracketing' prior assumptions and knowledge. In the above example, it would mean accepting that pairing in practice changes shape when the focus is on developing code rather than following a predefined procedure and that stand-up meetings may look different for different teams depending on their context. The research can then focus on how development is implemented in this context, what is the rationale for deviating from the textbook way of doing things and how have these practices developed.

This concern of preconceptions extends to assumptions and definitions of terms. Even in the same team, different members of the team might use terms differently from each other and from the researcher, e.g. the term 'prototype' is used in different disciplines in widely varying ways. It is therefore important to pay attention to the language team members use when discussing development and to challenge your preconceptions and assumptions on a regular basis to ensure that your understanding is correct. This allows the researcher to understand how and why methods and tools have been adapted and appropriated but also the wording developers use can provide insights into their point of view regarding technical matters or development processes. For example, how security incidents are referred to [36] or how development artefacts are talked about can indicate attitudes and concerns. The latter might also help the researcher understand team structure and responsibilities, e.g. the tech lead might talk about his or her design, whereas the newbie would talk about our design.

The importance of language is revisited below in Sect. 5 on analysis.

4.3 During the Study

Observation is key to ethnographic studies, and both participant and non-participant observation are legitimate forms of ethnography [21]. The data collected for observation is largely in the form of notes. Field notes are an important aspect of the work done by the ethnographer, and the researcher must be prepared to take

notes during the study. The question, though, is what notes to take? On the one hand, the researcher can document as many things as possible including the events, things people say, interruptions, event participants or documents being accessed or discussed, because at the start of the study, it's hard to know what will turn out to be significant. However, documenting everything will quickly become overwhelming. Textbooks sometimes recommend a template or schema for field notes. Such a schema can be very helpful and provide useful scaffolding at the start. Filling in the schema should not become the main purpose, though, and a tendency to just 'tick the boxes' needs to be resisted. The main purpose is, of course, to make sense of the team's work practices and to document what is relevant to that aim. An alternative to schemas is to use a framework of questions. There are simple frameworks such as 'who is there', 'where are they', 'what are they doing' and more detailed ones such as the following based on Robson and McCarten [47] (p. 328):

- Space: What is the physical space like, and how is it laid out?
- Actors: What are the names and relevant details of the people involved?
- Activities: What are the actors doing, and why?
- Objects: What physical objects are present, such as story cards?
- Acts: What are specific individual actions?
- Events: Is what you observe part of a special event?
- Time: What is the sequence of events?
- Goals: What are the actors trying to accomplish?
- Feelings: What is the mood of the group and of individuals?

These are only intended to help keep a focus on relevant issues so that the ethnographer is not overwhelmed, and they will evolve over time.

Today, many researchers use computers for their field notes, which makes it easier to link to other field material such as (sample) documents, photos and recordings. But handwritten notes are also popular, and some find them a better way to engage with their surroundings. Each researcher will need to find their own preference.

However, as mentioned in Sects. 3 and 4, the focus of the study will evolve and the field notes will reflect that. For instance, after noticing that the software architecture influenced the interactions between the software teams, de Souza and Redmiles [10] started to pay more attention to the inter-team interactions because they were sources of tension between the different product teams. As the focus changes, ethnographers also adopt different data collection techniques, for instance, during the field work, the researcher can decide to follow an artefact, e.g. a pull request, bug report, a presentation or the documentation of a piece of the software architecture. When this happens, the researcher should be prepared to ask for access to these artefacts, which may mean a physical (printout) or digital copy of it. By following an artefact, the researcher might shine light on a different set of new events, activities and/or informants who often will bring new perspectives into the field site being studied.

Last but not least, it is important to note that field notes will contain both objective information such as quotes, timestamps of events, links to recordings

and photos and subjective information, such as the researcher's reflections and interpretations of what is going on. Objective information is important, to anchor your emerging understanding of the work practices in actual observations. Verbatim quotations are extremely useful to present a credible report of the research. According to Fetterman [20], 'quotations allow the reader to judge the quality of the [ethnographic] work - how close the ethnographer is to the thoughts of natives in the field - and to assess whether the ethnographer used such data appropriately to support the conclusions'.

The subjective information includes the researchers' tentative interpretation of the events being observed, 'hypotheses' about what is being observed and questions the researcher asks themselves when collecting data. These reflections can evolve into important memos that in turn will develop into the parallel analysis of the data. It is, though, essential that the researcher clearly differentiates the subjective from the objective information in their field notes. This can be done by using different colours in a paper-based field diary or by splitting the page vertically or horizontally. Reflection and emerging hypothetical interpretations are, in qualitative research, validated in the field. For instance, during data collection, de Souza and Redmiles [57] started to wonder whether the phenomena observed would occur only with novice team members. To 'validate' whether this is the case, they decided to observe and interview senior team members. Through this, they found out that the phenomena was more general and occurred with team members of different experience.

In some cases, the events being observed will unfold so quickly that the researcher will not be able to take notes. In this case, ethnographers allocate time to record their memories in their field notes as soon as possible after the event and by the end of the day at the latest.

Nowadays, recording meetings is common and straightforward since some participants are online. In this case, it is important for the researcher to negotiate access to these recordings because they will be important data points for data analysis. They should be linked to the field diary so that the researcher later can connect them to the context in which they took place. However, being able to record meetings so easily is a potential disadvantage because it's easy to collect a huge amount of data, and there is a tendency for the researcher to lose concentration if they know data is being captured through other means.

In addition to observation, conducting ethnographic studies will likely employ a range of data collection methods including interviews and document analysis. Interviews are a chance to clarify meanings and the history of what has been observed. Such interviews might be formal or informal. Informal, or impromptu, interviews are often in situ and opportunistic; they will often occur in the context of an observation. In other words, the ethnographer, after witnessing an event that seems interesting, surprising or even expected, will use the opportunity to engage in a quick conversation with the informant to find out more details. These informal interviews will occur when opportunities arise, for instance, during a break or while waiting for a meeting. All this is done having in mind that the role of an ethnographer is to understand the member's point of view. Formal interviews require

an agenda and a negotiated date and time but may be the only opportunity to speak with some informants or to explore some kinds of issue.

In our experience, a good ethnographer strikes a balance between conducting formal and informal interviews. What this balance looks like depends on the context and the flexibility of the informants' working day. Too many informal interviews with the same informants can become irritating. Similarly, conducting formal interviews means asking people to allocate time for the conversation. However, checking information and making sure that observations and reflections are well founded are fundamental.

A good ethnographer also needs to pace themselves, plan their days carefully and be systematic about keeping records. This is particularly significant because data collection will go on for several days or even weeks, and it is easy for memories to fade or become confused. The researcher needs to be constantly reflecting about what data is being collected, including around events, conversations and artefacts, and revisit the emerging analysis concepts in light of the new observations. Ethnographers need to remember to avoid judging the work being observed and to keep focus on the informants' point of view, i.e. why they do the things they do. Being careful about data collection also means collecting data from different informants to gather different perspectives while, at the same time, not being a nuisance.

Finally, it is important to emphasise that ethnographic research, similar to other qualitative research methods, requires cycles of data collection and analysis. In other words, ethnographic research requires negotiating continuous and evolving access to the site. As discussed in the next section, during the data analysis, new foci might emerge and require new directions for data collection, such as access to new informants or new events or new documents. So the ethnographer must be continuously negotiating access to these during the research process.

4.4 Going Native

A key ability for an ethnographer is to be able to view activities as 'strange' so that they can question what is happening and uncover the significance of what is observed. But it's also key to be able to get inside a culture and elucidate the members' perspective. Keeping a balance between strangeness and familiarity can be challenging. If the balance tips too far in the direction of familiarity and it becomes hard to view things as strange, then the researcher is said to have 'gone native'.

Here regular debriefing meetings with supervisors and colleagues are important to keep the analytic distance.

The Educators' Corner: Pitfalls

- Becoming judgemental about observed practice, SO focus on why it is the way it is.
- 'Going native' meaning that the way informants behave and achieve things become 'normal', SO talk to other researchers and hold debriefings to help you regain the questioning mindset.
- Losing your 'strangeness' perspective and not knowing what needs to be told for others to understand your interpretations, SO try explaining what is happening to an alien.
- Human tendency is to gravitate towards people you like, but in the field it's important to attend to all informants, SO be aware of your own preferences and biases.
- It is easy to focus on a small number of informants because you become comfortable with them, SO make sure you choose a range of informants to get different perspectives, i.e. novices and experts, male and female, informants with different roles, etc.
- Observation can result in lots of data, some of which is not relevant, SO focus on what to pay attention to and choose carefully what data to collect.
- Assumptions or pre-conceived ideas can lead to misunderstandings, SO challenge your own assumptions and ask dumb questions to establish that your understanding is correct.
- Recording meetings and interviews is straightforward and easy but can result in huge amounts of data and loss of concentration, SO keep taking field notes even when the interactions are being recorded.

4.5 Leaving the Field

Eventually, the researcher will have to leave the host organisation or the open-source community because the software development project finishes, the research project ends or because the researcher has learned what could be learned from the ethnographic field work. Leaving the field should be done in a way that allows the researcher to come back to ask clarifying questions and open up for future collaboration on new mutually interesting research questions. It is therefore important that the researcher informs all of the informants that they are leaving the site and what will happen next with their data and its analysis.

Saying farewell to the team can be combined with a presentation of (preliminary) findings and insights that can also be used as an occasion for member checking. Member checking is an important aspect of any qualitative research (see also Sect. 5). It involves presenting the results of the research to the informants so that

they can 'validate' them. In ethnographic research, one way to do this is to formally present the results to the key informants who facilitated the ethnographer's access (see Sect. 4.1) or even to the whole development team. Another way is through informal presentations during the final days of data collection, such as during a break, while waiting for a meeting, etc. The format in which this is done varies depending on the team, project and overall context. On the one hand, feedback from the participants is important for the trustworthiness of the findings. On the other hand, presenting the findings can also serve as a 'thank you' for the team and can support the reflection and improvement of their own practices. If an organisation or team is interested enough in having a researcher on site, then they will also be interested to know what insights have been found. As one of our collaborators once said, 'the problem with running a project is that you're running, but having you here allowed us to stop and reflect'.

? Exercise 7: Prepare for the First Day for Your Field Work

1. Identify your key informant:
   Add to your study design a section motivating this and describing how the contact can be established.
2. Design your field diary:
   a. Decide whether to use an electronic diary or a paper diary. Decide on how to separate observations and reflections. Decide whether to use a schema for your field diary and, if so, which one.
   b. Add a few paragraphs to your study design motivating and presenting your choice.

! The Educators' Corner: Pitfalls

- Believing that you're in the field to improve practice, SO focus on understanding why it is the way it is.
- Having preconceived ideas such as for a new tool or process improvement, SO bracket your assumptions (see Sect. 4.2).
- Getting observations mixed up with your interpretations, SO structure your diary to make them distinct.
- Making too much of the detail of everyday practice rather than why things are the way they are, SO get into the habit of regularly asking yourself 'why'.

5 From Data to Text

The first part of the term Ethnography ('ethnos') means culture, while the second part ('graphy') refers to the writing, the presentation of information about the culture or practices observed. When using ethnography as a research method in software engineering, the presentation of the results is in the form of articles and less often in the form of a monograph. The challenge often is to reduce the rich experiences from the field into what can be presented in the format of a conference or journal submission. The more organised the ethnographer, the easier the task of analysing the collected data [21]. We first present the analysis process and thereafter discuss how to relate it to the software engineering discourse.

5.1 Reflective and Inductive Analysis

The analysis of ethnographic data is a reflective and inductive iterative process that starts in parallel to field work. It is possible and even recommended that ethnographers write temporary notes (often called memos) about what is observed in the field. The information captured in memos will be used backwards to identify where else a similar pattern showed up in the field work so far and forwards to inform the following period of data collection. For instance, as mentioned in the previous section, de Souza and Redmiles [57] interviewed experienced developers to find out whether their experience was similar to novice developers.

These memos are also an important starting point for a more formal and systematic analysis. But where to start the analysis is a conundrum when faced with so much rich data. One useful technique is to look for 'rich points' [2]. These are surprises or notable insights that researchers experience while in the field. Often, they may manifest as initial impressions when entering a new event or context (although any such impressions should be systematically validated). These may lead to narrative accounts of everyday practices or identifying patterns that appear across situations. The former is not simply a description of what was observed but includes a synthesis and interpretation of the data. Being careful with language both during analysis and in the ethnographic account is very helpful here: 'Certain birds can solve problems better than some dogs' carries a different meaning to 'birds are smarter than dogs'. In the process of capturing and reporting findings, it can be easy to summarise or abstract something that accidentally leaves out important context or nuance.

Ethnography, like other qualitative research methods, often employs coding techniques to identify patterns across situations, i.e. data are micro-analysed (line by line) to identify codes. When coding, an ethnographer is actually making sense of all the data collected. These codes are put together under a more abstract, high-order concept to explain what is going on [8, 30]. For instance, de Souza and Redmiles [57] created a code to indicate the concept of 'back merges': the fact

that a software developer had to incorporate code changes from other developers into their own workspace before integrating their code into the main codebase. Codes are created to minimise the number of elements that the researcher needs to consider. After creating codes, and often in parallel to them, codes can be grouped together to suggest a category, an even high-order concept. In this case, de Souza and Redmiles [57] created the category 'backward impact management' to indicate how a software developer continuously assesses the impact of the work performed by other developers on their own work and the appropriate actions developers adopted to avoid such impact. In other words, a 'back merge' (code) is a form of 'backward impact management' (category).

However, coding is not necessarily only informed by the data. For example, inductive codes might be combined with codes that relate observations to theoretical concepts. Likewise, there is no requirement for code coverage, as the data collection especially in the introduction of the observation is not yet focussed on what emerges to be the main theme. For a more detailed coverage of qualitative analysis, see the chapter by Treude in this volume and the further readings provided in Sect. 7.

In general, the goal of the analysis is to invoke all data collected (field notes, documents, code fragments, informal and formal interviews, etc.) to draw a clear picture of how the field site works, i.e. why the software developers do what they do. When envisioning this big picture, the ethnographer's attention will often wander through the data and will make logical leaps that lead to interesting insights. However, they must also 'backtrack to see whether the data will support these new ideas or invalidate them' [20]. Ethnographers and qualitative social scientists talk about a 'dialogue with the data'. During analysis, the researcher moves between formulating reflections and interpretations leading to codes, which then are tested with the empirical data [42]. This also includes data triangulation, i.e. exploring different sources of data and making sure they support or at least don't refute the result. For instance, the results from observation can be combined with the analysis of interviews or documents. In addition, an experienced ethnographer will use 'member checking', i.e. will share their writing, memos or any form of temporary results with trusted informants to gather feedback and find out whether their understanding of what is going on in the field site is recognised by the informants. It is important that the description of what was said and seen in the site be recognised by the informants, even if they do not fully agree with the interpretation of these results [20].

The Educators' Corner: Analysing Data Together

One way data analysis is taught in social sciences is to analyse data together. Here, researchers take an anonymised version of their own field material, an interview transcript, an instant messaging protocol or a video or audio recording and analyse it with others.

Of course it would not be possible to reconstruct the full ethnographic experience. However, in joint analysis, the teacher could point to relevant cues that connect the specific part of the field material to the broader context of the empirical work and that way illustrate how the brought picture gained through ethnographic field work allows them to then make sense of the specific document or conversation.

5.2 Writing Ethnography for Software Engineering Audiences: Reporting the Results

The results of analysis need to be related to software engineering literature to argue how the findings lead to new insights for the software engineering community. This has traditionally been difficult to achieve, as many software engineering researchers saw it as the goal of software engineering to improve industrial software development practice rather than to understand why practitioners do things the way they do. In recent years, though, this has started to change. Ethnography and qualitative empirical research are more accepted as a means to understand development practices better and hence to improve understanding which can then inform the development of new tools and techniques.

Although the overall theme of the research is likely to have been established before field work starts (if only to inform the choice of field site), the specific interesting aspects that support new insights for the software engineering community will evolve during field work and analysis. Because of this, the researcher might have to complement the usual initial literature study with a short field study to establish the value of their chosen research focus, especially for a M.Sc. thesis or PhD project.

Sometimes, additional analysis or even additional research is necessary to establish an argument for the relevance of the results. In this case, ethnography is used in combination with other studies using different techniques. An example of this is described in Plonka et al. [46]. In this research, ethnographic field work was complemented by video recordings of pair programming sessions. The analysis of ethnographic data led to the identification of a number of ways in which knowledge transfer takes place during pair programming. This in turn was supported by an in-depth interaction analysis of the video recordings, which both illustrated and deepened the understanding of knowledge transfer in pair programming and led to some practical recommendations [44].

An example where additional research developed into an empirical (sub)project in its own right is Unphon and Dittrich's article [63]. They present an interview study that was based on a finding in the above-mentioned ethnographic-grounded action research study. Unphon termed the observation of the technical lead communicating the architecture through one-on-one discussions 'walking architecture'. The interview study established that this practice was widespread in software product

development organisations and also uncovered some of the rationale behind these practices.

When writing up the study, the method section needs to state the basic information about the site and duration of the study. Also, a detailed account is needed of the field work methods, the analysis process and the measures implemented to assure the trustworthiness of the qualitative research. Often reviewers will ask for numbers, e.g. hours/days of observation, number of documents analysed and even the size of the system being developed by the team. These can be included to characterise the study itself, but do not be tempted to analyse these numbers quantitatively as part of the results. Also, the measures implemented to assure trustworthiness (see subsection above [47]) need to be presented in detail, and examples of how these methods were applied in this study will often be expected.

One dilemma is the conflict between the need to provide rich descriptions and the result orientation of the software engineering community. Passos et al. [41] also mention this as one of the challenges of ethnography in the context of software engineering. Rich descriptions allow the reader to follow the inductive reasoning that led to the findings that support the insights. Software engineering reviewers often find these rich descriptions too long and uninteresting. Here the qualitative researcher will have to strike a balance between the quality of the account and the requirements of the community. Using tables, charts or other techniques to summarise context and findings can help establish such a balance.

The Educators' Corner: Tips for Teaching

- Writing as a part of interpretation and reporting is not always comfortable for those with an SE background, so offer writing guidance.
- Language is important in the data, so include some analysis exercises that focus on the use of language.

! The Educators' Corner: Pitfalls

- First impressions can overshadow later observations, SO look for confirming AND disconfirming evidence.
- Succumbing to confirmation bias, SO seek confirming AND disconfirming evidence for your interpretations.
- Getting hung up on too much detail, SO sit back and consider the wider picture.
- Making too much of the detail of everyday practice rather than why things are the way they are, SO get into the habit of regularly asking yourself 'why'.

6 Ethnography and Research Ethics

As part of ethnographic field work, the researcher tries to become part of the real world environment that he or she researches. This means that the researcher will get to know about aspects of the team members' work and life that potentially can lead to harm when exposed in the wrong contexts. Reflection on research ethics therefore needs to be an integral part of ethnographic research.

In many universities, empirical research designs need to be approved by an ethics board. These boards might have diverging requirements from the considerations below. In some aspects, they might ask for more measures for assuring ethical implementation of the research design, while in other aspects the considerations below might appear more rigorous. Our discussion addresses what we think is needed to act ethically and responsibly as an ethnographic researcher in an industrial software engineering environment.

As empirical software engineering researchers, we often cooperate with project teams or IT departments that are part of a company. As both the cooperating team and the management can be expected to read our articles, the results of the ethnography in itself might lead to problems for the team or department in the organisation. Lucy Suchman argued in her seminal article 'Making Work Visible' [59] that the presentation of how work actually takes place in an industrial environment can, on its own, be a highly political act. However, the company itself can also face negative effects of the research.

In ethnographic research, ethics is not only about the effects of the research when published. How the researcher acts in the field and how their loyalties are perceived will affect the trust of the software team or project they are working with. How ethical issues are handled needs to be part of the planning of a research project.

The researcher needs to reflect on the impact of the research on different levels:

- The individual team member might face repercussions if it becomes visible that they do not work in adherence to the project's or the company's methods or policies.
- The software team could be affected by having their internal work practices exposed.
- The relationship between the team and management might suffer, if the team perceives the researcher as a control agent reporting to management.
- The company could suffer from exposure of internal work practices, e.g. with respect to the reputation and, through that, trust of customers and investors. Further, the exposure of the plans and development of innovative services might diminish the competitive advantage the company tried to achieve.

Although it is often possible to anonymise the company and the project to the wider audience and abstract from the specific software developed, the anonymisation of the identity of the team and of team members with respect to the company organisation might not be possible.

Handling your relationship with the individual developers, the team and the company in a responsible manner therefore needs to be addressed in a number of ways.

The researchers, one way or another, need to acquire informed consent. This means that team members are aware of the researcher and the research interest. They are aware of how the data collected will be used, who has access to it prior to publication and how the use of direct citations or observations will be communicated to them before disclosure inside the company or publication of research results. Informed consent might be achieved by asking the team members to sign a document. It also can be achieved by the researcher presenting themselves and detailing these topics, for example, at the beginning of a team meeting or through an e-mail.

Prior to publication and prior to the management clearing publication from a non-disclosure agreement (NDA) point of view, citations of team members and the presentation of their behaviour need to be member-checked with the individual team members. This has two purposes: to make sure that the researcher understood situations and behaviour in an adequate way and to allow the individual team members to control what is disclosed about their behaviour to management. To this end, it is common to create excerpts from the analysis section of publications or theses and share them with the relevant team members.

Open negotiation and communication about how findings are revealed to both the management and the academic public is crucial for trust between team members and researchers. In many cases, the company's management itself is not interested in being perceived as using researchers to 'spy' on their employees. In one of our projects, the management of the company took up the issue by asking in the beginning of the collaboration whether we researchers plan to act as a modern version of the 'time and motion' recorder. In a Tayloristic approach to the organisation of industrial production, researchers recorded the time it took for the individual operations of an assembly. The 'time and motion recorder' since then has become a symbol for control in many European countries. The discussion that followed this comment from our collaborators helped us clarify the role of the researchers. Further, it is important that the management is aware of the commitment of the researchers to not disclose findings if the employees and the team disagree.

In some cases, especially when combining ethnographic research with action and design research, the researchers might consider co-authoring research results together with (some of) the involved practitioners. This requires the contents, conclusion and structure of the publication to be discussed with the relevant practitioners. During the writing process, the researchers should then as much as possible involve the practitioners so that all authors agree on the text, especially the findings and conclusions.

Non-disclosure agreements and contracts handling intellectual property detail the procedures to make sure the interests of all parties are considered before results are published. Most collaboration agreements between companies and universities include procedures for review and acceptance of publications containing the research results. However, we recommend taking up this topic with the company contact early and discussing any aspects of the project that can become critical for the company if disclosed. Some companies might prefer to not be mentioned by name in the article, although a sufficient description of the research context may result in identification anyway, e.g. describing the collaborator as 'a large multi-national telecomms company based in Finland' suggests only a few possible companies. Sometimes, certain aspects of the software under development and the project could provide information competitors could use to harm the company. In most cases, the researchers can find ways to disguise critical information.

Ethnographic research is sometimes used as a first step in action or design research. In such cases, we recommend extending the established practices to handle relationships with the developers, the team and the company to the other research activity. Dittrich et al. [17] propose designing their participatory action research approach in a way to keep the members' perspective throughout the action research cycle.

? Exercise 8: Consider Research Ethics

Review your ethnographic study design from a research ethics point of view, and revise it based on the issues discussed in this section.

7 Final Comments and Further Reading

At the start of this chapter, we provided a simple answer as to why ethnography is different to other qualitative methods. Having read through the chapter and hopefully completed the exercises along the way, we hope that you have a clearer and more holistic answer to this question. Ethnography is characterised by focusing on the members' point of view and understanding why things are the way they are. Understanding current practice helps inform decisions about future practice, processes and tools. But it's not the specific practice that is relevant; it's the significance of that practice. Ethnography is good at answering 'how' and 'why' questions and makes certain demands of the researcher, such as focusing on everyday detail and writing thick descriptions. It has been used to address research questions in four contexts: investigations of social and human aspects, to inform

tool design, to inform process change and to complement other research methods. So we invite you to consider this question again, this time for yourselves:

? Exercise 9: Ethnography and Other Research Methods

1. What are the key differences between ethnography and:
   a. Grounded theory
   b. Action research
   c. Case study
2. How might an ethnographic study complement a research programme that involves:
   a. Experiments
   b. Survey
   c. Systematic literature review

The following are suggestions for further reading on this topic:

Fetterman, D.M.: Ethnography: Step-by-step. Sage Publications (2020)
This is a very practical book about ethnography, and although written with a social science audience in mind, it is accessible to readers from a wide range of backgrounds.

Hammersley, M., Atkinson, P.: Ethnography: Principles in Practice. Routledge (2019)
This book is written from a social sciences perspective and aims to take an objective view of ethnography and its limitations. It deepens and problematises some of the issues raised in this chapter and introduces some others.

Pink, S., Horst, H., Postill, J., Hjorth, L., Lewis, T., Tacchi, J.: Digital Ethnography: Principles and Practice. SAGE Publications Ltd, United Kingdom (2016)
This text provides an introduction to digital ethnography and how different media and technologies affect the ethnographic research endeavour, both through their effects on the world being studied and the opportunities that technology provides for the researcher.

Randall, D.W., Harper, R.H.R., Rouncefield, M.: Fieldwork for Design: Theory and Practice. Computer Supported Cooperative Work. Springer (2007)
The book presents a social science perspective on ethnography in the context of computer-supported cooperative work. It explains ethnography and how it can be brought to bear on design of software to support cooperation. Though this is different from using ethnography in software engineering, it supports the possibility to use ethnography in different disciplinary contexts and reflects on the challenges of doing so.

Zhang, H., Huang, X., Zhou, X., Huang, H., Babar, M.A.: Ethnographic Research in Software Engineering: A Critical Review and Checklist. In: Proceedings ESEC/FSE 2019, Tallinn, Estonia, August 26–30, 2019, pp. 659–670. ACM (2019)
As research is progressing, new ethnographic studies are published continuously. This is one recent literature review.

Sharp, H., Dittrich, Y., de Souza, C.R.: The Role of Ethnographic Studies in Empirical Software Engineering. IEEE Transactions on Software Engineering 42(8), 786–804 (2016)
This article is a companion to this chapter. It introduces the role of ethnography in empirical software engineering through a set of ethnographic studies that have been conducted with software engineering as their focus.

Acknowledgments Thanks to our colleagues and students who have joined us in our daring endeavor into researching everyday practices of software engineering. Together with you, we learned what was challenging in ethnographic studies. Thanks to the software engineers who collaborated with us. You have taught us so much about how software engineering actually takes place. Thanks also to the reviewers and editors of this volume for your constructive feedback and support.

References

1. Agar, M.: Independents Declared: The Dilemmas of Independent Trucking. Smithsonian Series in Ethnographic Inquiry (1986)
2. Agar, M.H.: The professional stranger: An informal introduction to ethnography. Academic Press, San Diego (1996). papers3://publication/uuid/96F108B0-0BDC-4D1B-AAF2-BB963B9F08E9. https://www.citeulike.org/group/268/article/221525
3. Begum, M.: On bug resolution: Artefacts, information flow and triage. Master's thesis, IT University of Copenhagen (2020)
4. Bentley, R., Hughes, J.A., Randall, D., Rodden, T., Sawyer, P., Shapiro, D., Sommerville, I.: Ethnographically-informed systems design for air traffic control. In: Proceedings of the 1992 ACM Conference on Computer-Supported Cooperative Work, pp. 123–129 (1992)
5. Blomberg, J., Suchman, L., Trigg, R.H.: Reflections on a work-oriented design project. Human Comput. Interact. **11**(3), 237–265 (1996)
6. Capiluppi, A., Fernandez-Ramil, J., Higman, J., Sharp, H., Smith, N.: An empirical study of the evolution of an agile-developed software system. In: 29th International Conference on Software Engineering (ICSE'07), pp. 511–518. IEEE (2007)
7. Cetina, K.K.: Objectual Practice. In: The Practice Turn in Contemporary Theory. Routledge (2005)
8. Clarke, V., Braun, V.: Thematic Analysis: A Practical Guide. SAGE Publications Ltd (2022)
9. De Souza, C.R., Redmiles, D., Mark, G., Penix, J., Sierhuis, M.: Management of interdependencies in collaborative software development. In: 2003 International Symposium on Empirical Software Engineering, 2003. ISESE 2003. Proceedings., pp. 294–303. IEEE (2003)
10. De Souza, C.R., Redmiles, D.F.: The awareness network, to whom should I display my actions? and, whose actions should I monitor? IEEE Trans. Software Eng. **37**(3), 325–340 (2011)

11. Deshpande, A., Sharp, H., Barroca, L., Gregory, P.: Remote working and collaboration in agile teams. In: Ågerfalk, P.J., Levina, N., Kien, S.S. (eds.) Proceedings of the International Conference on Information Systems - Digital Innovation at the Crossroads, ICIS 2016, Dublin, Ireland, December 11–14, 2016. Association for Information Systems (2016). http://aisel.aisnet.org/icis2016/ManagingIS/Presentations/12
12. Dingsøyr, T., Bjørnson, F.O., Schrof, J., Sporsem, T.: A longitudinal explanatory case study of coordination in a very large development programme: the impact of transitioning from a first-to a second-generation large-scale agile development method. Empir. Softw. Eng. **28**(1), 1 (2023)
13. Dittrich, Y.: Doing Empirical Research on Software Development: Finding a Path between Understanding, Intervention, and Method Development, pp. 243–262. The MIT Press (2002). https://doi.org/10.7551/mitpress/6308.003.0016
14. Dittrich, Y.: What does it mean to use a method? towards a practice theory for software engineering. Inf. Softw. Technol. **70**, 220–231 (2016). https://doi.org/10.1016/j.infsof.2015.07.001
15. Dittrich, Y., Lindeberg, O.: How use-oriented development can take place. Inf. Softw. Technol. **46**(9), 603–617 (2004). https://doi.org/10.1016/j.infsof.2003.11.002
16. Dittrich, Y., Michelsen, C.B., Tell, P., Lous, P., Ebdrup, A.: Exploring the evolution of software practices. In: Devanbu, P., Cohen, M.B., Zimmermann, T. (eds.) ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8–13, 2020, pp. 493–504. ACM (2020). https://doi.org/10.1145/3368089.3409766
17. Dittrich, Y., Rönkkö, K., Eriksson, J., Hansson, C., Lindeberg, O.: Cooperative method development. Empir. Softw. Eng. **13**(3), 231–260 (2008). https://doi.org/10.1007/s10664-007-9057-1
18. Edwards, C.: Making paper strips digital (2017). https://nats.aero/blog/2017/08/making-paper-strips-digital/. Last accessed: 13. 06. 2024
19. Engeström, Y., et al.: Activity theory and individual and social transformation. Perspect. Activity Theory **19**(38), 19–30 (1999)
20. Fetterman, D.M.: Ethnography: Step-by-Step. Sage Publications (2010)
21. Fetterman, D.M.: Ethnography: Step-by-Step. Sage Publications (2020)
22. Gearing, R.E.: Bracketing in research: A typology. Q. Health Res. **14**(10), 1429–1452 (2004). https://doi.org/10.1177/1049732304270394. PMID: 15538009
23. Giuffrida, R.: How social software supports communicative and coordinative practices in global software development. Ph.D. thesis, IT university of Copenhagen (2014)
24. Giuffrida, R., Dittrich, Y.: How social software supports cooperative practices in a globally distributed software project. In: Sharp, H., Prikladnicki, R., Begel, A., de Souza, C.R.B. (eds.) Proceedings of the 7th International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE 2014, Hyderabad, India, June 2–3, 2014, pp. 24–31. ACM (2014). https://doi.org/10.1145/2593702.2593716
25. Giuffrida, R., Dittrich, Y.: A conceptual framework to study the role of communication through social software for coordination in globally-distributed software teams. Inf. Softw. Technol. **63**, 11–30 (2015). https://doi.org/10.1016/j.infsof.2015.02.013
26. Hammersley, M., Atkinson, P.: Ethnography: Principles in Practice. Routledge (2019)
27. Heath, C., Luff, P.: Collaborative activity and technological design: Task coordination in London underground control rooms. In: Bannon, L.J., Robinson, M., Schmidt, K. (eds.) Proceedings of the Second European Conference on Computer Supported Cooperative Work, 24–27 September 1991, Amsterdam, The Netherlands, pp. 65–80. Kluwer (1991). https://doi.org/10.1007/978-94-011-3506-1_5
28. Herbsleb, J.D., Grinter, R.E.: Splitting the organization and integrating the code: Conway's law revisited. In: Boehm, B.W., Garlan, D., Kramer, J. (eds.) Proceedings of the 1999 International Conference on Software Engineering, ICSE' 99, Los Angeles, CA, USA, May 16–22, 1999, pp. 85–95. ACM (1999). https://doi.org/10.1145/302405.302455

29. Hine, C.: Virtual Ethnography: Modes, Varieties, Affordances. The SAGE Handbook of Online Research Methods, pp. 257–270 (2008)
30. Hoda, R.: Qualitative Research with Socio-Technical Grounded Theory: A Practical Guide to Qualitative Data Analysis and Theory Development in the Digital World. Springer (2024)
31. Hutchins, E.: Cognition in the Wild. MIT Press (1995)
32. Jordan, B.: Cosmopolitical obstetrics: Some insights from the training of traditional midwives. Soc. Sci. Med. **28**(9), 925–937 (1989)
33. Kozinets, R.V.: Netnography: The Essential Guide to Qualitative Social Media Research, 3rd edn. SAGE Publications Ltd (2019)
34. Lave, J., Wenger, E.: Situated Learning: Legitimate Peripheral Participation. Cambridge University Press (1991). https://doi.org/10.1017/CBO9780511815355
35. Lopez, T., Sharp, H., Bandara, A.K., Tun, T., Levine, M., Nuseibeh, B.: Security responses in software development. ACM Trans. Softw. Eng. Methodol. **32**(3), 64:1–64:29 (2023). https://doi.org/10.1145/3563211
36. Lopez, T., Tun, T.T., Bandara, A.K., Levine, M., Nuseibeh, B., Sharp, H.: Taking the middle path: Learning about security through online social interaction. IEEE Softw. **37**(1), 25–30 (2020). https://doi.org/10.1109/MS.2019.2945300
37. Low, J., Johnson, J., Hall, P.A.V., Hovenden, F., Rachel, J., Robinson, H., Woolgar, S.: Read this and change the way you feel about software engineering. Inf. Softw. Technol. **38**(2), 77–87 (1996). https://doi.org/10.1016/0950-5849(95)01046-7
38. Mackay, W.E.: Is paper safer? the role of paper flight strips in air traffic control. ACM Trans. Comput. Hum. Interact. **6**(4), 311–340 (1999). https://doi.org/10.1145/331490.331491
39. Martin, D.B., Rooksby, J., Rouncefield, M., Sommerville, I.: 'good' organisational reasons for 'bad' software testing: An ethnographic study of testing in a small software company. In: 29th International Conference on Software Engineering (ICSE 2007), Minneapolis, MN, USA, May 20–26, 2007, pp. 602–611. IEEE Computer Society (2007). https://doi.org/10.1109/ICSE.2007.1
40. Orr, J.E.: Talking About Machines: An Ethnography of a Modern Job. Cornell University (1990)
41. Passos, C., Cruzes, D.S., Dybå, T., Mendonça, M.G.: Challenges of applying ethnography to study software practices. In: Runeson, P., Höst, M., Mendes, E., Andrews, A.A., Harrison, R. (eds.) 2012 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '12, Lund, Sweden - September 19–20, 2012, pp. 9–18. ACM (2012). https://doi.org/10.1145/2372251.2372255
42. Pawluch, D., McLuhan, A., Shaffir, W.: Doing Ethnography. In: Routledge Handbook of Pragmatics. Routledge (2017)
43. Pink, S., Horst, H., Postill, J., Hjorth, L., Lewis, T., Tacchi, J.: Digital Ethnography: Principles and Practice. SAGE Publications Ltd, United Kingdom (2016)
44. Plonka, L.: Unpacking collaboration in pair programming in industrial settings. Ph.D. thesis, Open University, Milton Keynes, UK (2012). https://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.577973
45. Plonka, L., Segal, J., Sharp, H., van der Linden, J.: Collaboration in pair programming: Driving and switching. In: Sillitti, A., Hazzan, O., Bache, E., Albaladejo, X. (eds.) Agile Processes in Software Engineering and Extreme Programming - 12th International Conference, XP 2011, Madrid, Spain, May 10–13, 2011. Proceedings, Lecture Notes in Business Information Processing, vol. 77, pp. 43–59. Springer (2011). https://doi.org/10.1007/978-3-642-20677-1_4
46. Plonka, L., Sharp, H., van der Linden, J., Dittrich, Y.: Knowledge transfer in pair programming: An in-depth analysis. Int. J. Hum. Comput. Stud. **73**, 66–78 (2015). https://doi.org/10.1016/j.ijhcs.2014.09.001
47. Robson, C., McCartan, K.: Real World Research: A Resource for Users of Social Research Methods in Applied Settings. Wiley, Hoboken (2016)
48. Rogers, Y., Sharp, H., Preece, J.: Interaction Design: Beyond Human-Computer Interaction, 6th edn. Wiley (2023). https://www.id-book.com/

49. Rönkkö, K., Dittrich, Y., Randall, D.W.: When plans do not work out: How plans are used in software development projects. Comput. Support. Cooperative Work. **14**(5), 433–468 (2005). https://doi.org/10.1007/s10606-005-9004-x
50. Schatzki, T.R.: Social Practices: A Wittgensteinian Approach to Human Activity and the Social. Cambridge University Press, New York (1996)
51. Sharp, H., Dittrich, Y., de Souza, C.R.: The role of ethnographic studies in empirical software engineering. IEEE Trans. Software Eng. **42**(8), 786–804 (2016)
52. Sharp, H., Giuffrida, R., Melnik, G.: Information flow within a dispersed agile team: A distributed cognition perspective. In: Wohlin, C. (ed.) Agile Processes in Software Engineering and Extreme Programming - 13th International Conference, XP 2012, Malmö, Sweden, May 21–25, 2012. Proceedings, Lecture Notes in Business Information Processing, vol. 111, pp. 62–76. Springer (2012). https://doi.org/10.1007/978-3-642-30350-0_5
53. Sharp, H., Robinson, H.: An ethnographic study of XP practice. Empir. Softw. Eng. **9**(4), 353–375 (2004). https://doi.org/10.1023/B:EMSE.0000039884.79385.54
54. Sharp, H., Robinson, H.: Collaboration and co-ordination in mature extreme programming teams. Int. J. Hum. Comput. Stud. **66**(7), 506–518 (2008). https://doi.org/10.1016/j.ijhcs.2007.10.004
55. Sharp, H., Robinson, H., Petre, M.: The role of physical artefacts in agile software development: Two complementary perspectives. Interact. Comput. **21**(1–2), 108–116 (2009)
56. Sharp, H., Robinson, H., Woodman, M.: Software engineering: Community and culture. IEEE Softw. **17**(1), 40–47 (2000). https://doi.org/10.1109/52.819967
57. de Souza, C.R., Redmiles, D.F.: An empirical study of software developers' management of dependencies and changes. In: Proceedings of the 30th International Conference on Software Engineering, pp. 241–250 (2008)
58. Strode, D.E., Sharp, H., Barroca, L., Gregory, P., Taylor, K.: Tensions in organizations transforming to agility. IEEE Trans. Eng. Manag. **69**(6), 3572–3583 (2022). https://doi.org/10.1109/TEM.2022.3160415
59. Suchman, L.A.: Making work visible. Commun. ACM **38**(9), 56–64 (1995). https://doi.org/10.1145/223248.223263
60. Tell, P., Babar, M.A.: Activity theory applied to global software engineering: Theoretical foundations and implications for tool builders. In: 2012 IEEE Seventh International Conference on Global Software Engineering, Porto Alegre, Rio Grande do Sul, Brazil, August 27–30, 2012, pp. 21–30. IEEE Computer Society (2012). https://doi.org/10.1109/ICGSE.2012.24
61. Unphon, H.: Re-engineering for evolvability: Considering social as well as technical requirements for software products. Ph.D. thesis, IT-Universitetet i København, Denmark (2010)
62. Unphon, H., Dittrich, Y.: Organisation matters: How the organisation of software development influences the development of product line architecture. In: Software Engineering, SE 2008, February 12–14, 2008 Innsbruck, Austria, pp. 178–183. ACTA Press, Canada (2008). Er den peer-reviewed ? Ja.; Software Engineering, SE 2008, as part of the 26th IASTED International Multi-Conference on APPLIED INFORMATICS ; Conference date: 12-02-2008 Through 14-02-2008
63. Unphon, H., Dittrich, Y.: Software architecture awareness in long-term software product evolution. J. Syst. Softw. **83**(11), 2211–2226 (2010). https://doi.org/10.1016/j.jss.2010.06.043
64. Ybema, S., Yanow, D., Wels, H., Kamsteeg, F. (eds.): Organizational Ethnography: Studying the Complexities of Everyday Life. SAGE Publications Ltd (2009)
65. Zaina, L.A.M., Sharp, H., Barroca, L.: UX information in the daily work of an agile team: A distributed cognition analysis. Int. J. Hum. Comput. Stud. **147**, 102574 (2021). https://doi.org/10.1016/j.ijhcs.2020.102574