The Evolution of Empirical Methods in Software Engineering

Michael Felderer, Guilherme Horta Travassos

Abstract Empirical methods like experimentation have become a powerful means to drive the field of software engineering by creating scientific evidence on software development, operation, and maintenance, but also by supporting practitioners in their decision making and learning. Today empirical methods are fully applied in software engineering. However, they have developed in several iterations since the 1960s. In this chapter we tell the history of empirical software engineering and present the evolution of empirical methods in software engineering in five iterations, i.e., (1) mid-1960s to mid-1970s, (2) mid-1970s to mid-1980s, (3) mid-1980s to end of the 1990s, (4) the 2000s, and (5) the 2010s. We present the five iterations of the development of empirical software engineering mainly from a methodological perspective and additionally take key papers, venues, and books, which are covered in chronological order in a separate section on recommended further readings, into account. We complement our presentation of the evolution of empirical software engineering by presenting the current situation and an outlook in Sect. 4 and the available books on empirical software engineering Furthermore, based on the chapters covered in this book we discuss trends on contemporary empirical methods in software engineering related to the plurality of research methods, human factors, data collection and processing, aggregation and synthesis of evidence, and impact of software engineering research.

Michael Felderer
University of Innsbruck, Austria,
Blekinge Institute of Technology, Sweden
e-mail: michael.felderer@uibk.ac.at

Guilherme Horta Travassos
PESC/COPPE - Federal University of Rio de Janeiro, Brazil,
CNPq Researcher
e-mail: ght@cos.ufrj.br

1 Introduction

The term software engineering originated in the early 1960s (Hey et al., 2014). During the NATO Software Engineering Conferences held in 1968 and 1969, participants made explicit that engineering software requires dedicated approaches that are separate from those for the underlying hardware systems. Until that "software crisis", software-related research mostly focused on theoretical aspects, e.g., algorithms and data structures used to write software systems, or practical aspects, e.g., an efficient compilation of software for particular hardware systems (Guéhéneuc and Khomh, 2019). Since then, these topics are investigated in computer science, which pertains to understanding and proposing theories and methods related to the efficient computation of algorithms, and differs from software engineering (research), which has become a very dynamic discipline on its own since its foundation in the 1960s. IEEE (1990, 2010) defines software engineering (SE) as: (1) The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; that is, the application of engineering to software, and (2) The study of approaches as in (1). Software engineering also differs from other engineering disciplines due to the immaterial nature of software not obeying physical laws and the importance of human factors as software is written by people for people. Software engineering is fundamentally an empirical discipline, where knowledge is gained applying direct and indirect observation or experience. Approaches to software development, operation, and maintenance must be investigated by empirical means to be understood, evaluated, and deployed in proper contexts. Empirical methods like experimentation are therefore essential in software engineering to gain scientific evidence on software development, operation, and maintenance, but also to support practitioners in their decision making and learning (Travassos et al., 2008). The application of empirical methods makes software engineering more objective and less imprecise, facilitating the transfer of software technologies to the industry (Shull et al., 2001). Software engineers learn by observing, exploring, and experimenting. The level of learning depends on the degree of observation or intervention (Thomke, 2003) promoted by the experiences and studies performed.

Traditionally, *empirical software engineering* (ESE) is the area of research that emphasizes the use of empirical methods in the field of software engineering. According to Harrison and Basili (1996), "Empirical software engineering is the study of software-related artifacts for the characterization, understanding, evaluation, prediction, control, management, or improvement through qualitative or quantitative analysis. The quantitative studies may range from controlled experimentation to case studies. Qualitative studies should be well defined and rigorous." The role and importance of the different types of empirical methods in software engineering have evolved since the foundation of software engineering. In this book chapter, we discuss the evolution of empirical methods in software engineering and especially also take key venues and books into account as they reflect that evolution.

The chapter is organized as follows. In Sect. 2, we provide background on empirical research methods in software engineering. In Sect. 3, we present the evolution of empirical software engineering by describing five iterations of its development.

Based on that "historical" perspective on empirical software engineering, in Sect. 4 we describe current trends in empirical software engineering based on the chapters on contemporary empirical methods in software engineering covered in this book. In Sect. 5, we present the available books on empirical methods in software engineering in chronological order as recommended further reading. Finally, in Sect. 6, we conclude this chapter.

2 Empirical Research Methods in Software Engineering

The scientific approach typically consists of observation, measurement, and experimentation. Observation helps researchers to formulate essential questions about a phenomenon under study to build models and to derive hypotheses that can be tested through experimentation. Measurement is essential for both observation and experimentation. A scientific hypothesis must be refutable to be meaningfully tested. Tested hypotheses are compiled and communicated in the form of laws or theories. At the heart of the scientific approach are research methods in general and the empirical method in particular. Empirical methods leverage evidence obtained through observation, measurement, or experimentation to address a scientific problem. Evidence should be based on qualitative and quantitative research. In this section, we provide an overview of research methods in software engineering in general and empirical methods in particular.

2.1 Research Methods

To perform scientific research in software engineering, one has to understand the available research methods and their limitations. For the field of software engineering, Basili (1993) and Glass (1994) summarized four research methods: scientific, engineering, empirical, and analytical.

The so-called *scientific method* observes the world and builds a model based on the observations, e.g., a simulation model of the software process or product. The scientific method is inductive and tries to extract from the world some model that can explain a phenomenon and to evaluate whether the model is representative for the phenomenon under observation. It is a model-building approach.

The *engineering method* studies current solutions, proposes changes, and then evaluates them. It suggests the most appropriate solutions, develops, measures and analyzes, and repeats until no further improvement is possible. It is an evolutionary improvement-oriented approach that assumes the existence of some model of the software process or product. It modifies this model to improve the objects of study.

The *empirical method* proposes a model, and evaluates it through empirical studies like case studies or experiments. The empirical method normally follows an iterative and incremental approach that can begin with an exploratory survey, followed by

case studies in an industrial context to better understand specific phenomena and controlled experiments to investigate cause-effect relationships.

The *analytical method* proposes a formal theory, develops the theory, derives the results, and, if possible, compares it with empirical observations. It is deductive and provides an analytical basis for developing a model.

Traditionally, the analytical method is used in the more formal areas of electrical engineering and computer science, but is important for software engineering as well, e.g., when building mathematical models for software reliability growth (Lyu et al., 1996). The scientific method, inspired by natural science, is traditionally used in applied areas, such as the simulation of a sensors network to evaluate its performance. However, simulations are used as a means for conducting an experiment as well (Baros et al., 2004). The engineering method is dominating in industry (Wohlin et al., 2012). The empirical method, mainly using empirical strategies, has traditionally been used in social sciences and psychology, where one is unable to state any laws of nature but concerned with human behavior. The engineering and the empirical method can be seen as variations of the scientific method (Basili, 1993). This overlap and an integrated view of the scientific, engineering, and empirical methods is also an underlying design principle of this book on empirical methods. It considers not only chapters on traditional empirical strategies like surveys, but for instance, also a chapter on simulation-based studies, which are closer to the scientific method as defined above, or a chapter on design science, which can tightly be linked to the engineering method. All of these investigation strategies refer to empirical methods.

2.2 Empirical Methods

Empirical methods rely on the collected data. Data collection methods may involve qualitative or quantitative data. Some widely used *qualitative data collection methods* in software engineering are interviews and participant observation (Seaman, 1999). Some commonly used *quantitative data collection methods* are archival data, surveys, experiments, and simulation (Wohlin et al., 2012). Once data are collected, the researcher needs to analyze the data by using *qualitative analysis methods*, e.g., grounded theory, thematic analysis or hermeneutics, and *quantitative analysis methods*, e.g., statistical analysis and mathematical modeling approaches.

In general, there are three widely recognized research processes called quantitative research, qualitative research, and semi-quantitative research. An alternative option is the combination of both qualitative and quantitative research, denoted as mixed research (Creswell and Creswell, 2018). The distinction between qualitative and quantitative research comes not only from the type of data collected, but also the objectives, types of research questions posed, analysis methods, and the degree of flexibility built into the research design as well (Wohlin and Aurum, 2015). *Qualitative research* aims to understand the reason (i.e., "why") and mechanisms (i.e., "how") explaining a phenomenon. A popular method of qualitative

research is case study research, which examines a set of selected samples in detail to understand the phenomenon illustrated by the samples. For instance, a qualitative study can be conducted to understand the impediments of automating system tests. *Quantitative research* is a data-driven approach used to gain insights about an observable phenomenon. Data collected from observations are analyzed using mathematical and statistical models to derive quantitative relationships between different variables capturing different aspects of the phenomenon under study. A popular method of quantitative research are controlled experiments to examine cause-effect relationships between different variables characterizing a phenomenon in a controlled environment. For instance, different review techniques could be compared via a controlled experiment. *Mixed research* collects quantitative and qualitative data. It is a particular form of *multi-method research*, which combines different research methods to answer some hypotheses, and is often used in empirical software engineering due to the lack of theories in software engineering with which we interpret quantitative data and due to the need to discuss qualitatively the impact of the human factor on any experiments in software engineering (Guéhéneuc and Khomh, 2019). *Semi-quantitative research* deals with approximate measurements to data rather than exact measurements (Bertin, 1978). It looks for understanding the behavior of a system based on causal relations between the variables describing the system. Semi-quantitative models allow one to express what is known without making inappropriate assumptions, simulating ranges of behavior rather than values of point (Widman, 1989). It has many applications in both the natural and social sciences. Semi-quantitative research supports cases where direct measurements are not possible, but where it is possible to estimate an approximated behavior. In other words, this type of study is applied in scenarios where the numerical values in the mathematical relations governing the changes of a system are not known. In this context, the direction of change is known, but not the size of its effect (Ogborn and Miller, 1994). Simulation-based studies in software engineering can benefit from using semi-quantitative research (Araújo et al., 2012).

The three major and well-established empirical methods in software engineering are: survey, case study, and experiment (Wohlin et al., 2012). Primary studies using such methods can be performed in-vivo, in-vitro, in-virtuo, and in-silico (Travassos and Barros, 2003). *In-vivo* studies involve participants and projects in their natural environments and contexts. Usually executed in software development organizations throughout the software development process under real working conditions. *In-vitro* studies are performed in controlled environments, such as laboratories or controlled communities, under configured working conditions. *In-virtuo* studies have the subjects interacting with a computerized model of reality. The behavior of the environment with which subjects interact is described as a model and represented by a computer program. *In-silico* studies represent both subjects and real world as computer models. The environment is fully composed of computer models to which human interaction is reduced to a minimum.

A *survey* is a system for collecting information from or about subjects (people, projects, among others) to describe, compare, or explain their knowledge, attitudes, and behavior (Fink, 2003). A survey is often an investigation performed in retrospect,

when, for instance, a tool or technique, has been in use for a while (Pfleeger, 1995). The primary means of gathering qualitative or quantitative data are interviews or questionnaires. These are done through taking a sample that is representative of the population to which is generalized.

A *case study* in software engineering is an empirical inquiry that draws on multiple sources of evidence to investigate one or a small number of instances of a contemporary software engineering phenomenon within its real-life context, especially when the boundary between phenomenon and context cannot be clearly specified (Runeson et al., 2012).

An *experiment* is used to examine cause-effect relationships between different variables characterizing a phenomenon (Guéhéneuc and Khomh, 2019). Experiments allow researchers to verify, refute, or validate hypotheses formulated about the phenomenon under study. In a controlled experiment, one variable of the study setting is manipulated, and based on randomization, different treatments are applied to or by different subjects while keeping other variables constant, and measuring the effects on outcome variables (Wohlin et al., 2012). A quasi-experiment is similar to a controlled experiment, where the assignment of treatments to subjects cannot be based on randomization, but emerges from the characteristics of the subjects or objects themselves (Wohlin et al., 2012). Replication experiments reproduce or quasi-reproduce previous experiments with the objectives to confirm or infirm the results from previous experiments or to contrast previous results in different contexts (Guéhéneuc and Khomh, 2019).

Regardless of the applied empirical method, to acquire scientific evidence about the investigated software engineering phenomena involves observation, measurement, and experimentation of the world and existing solutions. It demands the proposition of models and theories describing the observed behavior, collecting and analyzing data, putting the hypotheses under proof, and repeating the overall process over time to strengthen the evidence on the observed phenomena. Based on several primary studies, in which direct observations and measurements about the objects of interest are made, whether by surveys, experiments, or case studies, which are there also called *empirical strategies*, one can perform secondary studies. A *secondary study* does not generate any data from direct observation or measurement, instead, it analyses a set of primary studies and usually seeks to aggregate the results from these to provide stronger forms of evidence about a particular phenomenon (Kitchenham et al., 2015). Secondary studies typically appear as *systematic (literature) reviews*, which aim to provide an objective and unbiased approach to finding relevant primary studies, and for extracting, aggregating and synthesizing the data from these (Kitchenham et al., 2015). A particular type of a systematic review is a *systematic mapping study* (Petersen et al., 2015), which classifies studies to identify clusters of studies (that could form the basis of a fuller review with more synthesis) and gaps indicating the need for more primary studies.

The scientific or industrial significance of empirical studies depends on their validity, i.e., the degree to which one can trust the outcomes of an empirical study (Kitchenham et al., 2015). Validity is usually assessed in terms of four commonly encountered forms of *threats to validity*: internal, external, construct, and

conclusion validity (Shadish et al., 2002). *Internal validity* refers to inferences that the observed relationship between treatment and outcome reflects a cause-effect relationship. *External validity* refers to whether a cause-effect relationship holds over other conditions, including persons, settings, treatment variables, and measurement variables. *Construct validity* refers to how concepts are operationalized as experimental measures. *Conclusion validity* refers to inferences about the relationship between treatment and outcome variables.

The accomplishment of empirical studies relies on performing well-defined and evolutionary activities. The classical empirical study process consists of five phases: definition, planning, operation, analysis, and interpretation, as well as reporting and packaging (Juristo and Moreno, 2001; Malhotra, 2016). The definition phase makes the investigated problem and overall objectives of the study explicit. The planning phase covers the study design and includes the definition of research questions and hypotheses as well as the definition of data collection, data analysis, and validity procedures. In the operation phase, the study is actually conducted. In the analysis and interpretation phase, the collected data is analyzed, assessed, and discussed. Finally, in the reporting and packaging phase, the results of the study are reported (e.g., in a journal article, a conference paper, or a technical report) and suitably packaged to provide study material and data. The latter has become more critical recently due to the open science movement.

3 Evolution of Empirical Software Engineering

The application of empirical methods in general and empirical software engineering, in particular, are well established in software engineering research. Almost all papers published in major software engineering venues these days include an empirical study (Theisen et al., 2017). Furthermore, since 2000, research methodology has received considerable attention in the software engineering research community resulting in many available publications on empirical research methodology in software engineering. In a recent mapping study, Molléri et al. (2019) identified 341 methodological papers on empirical research in software engineering.

The application of empirical methods and the underlying research methodology has developed iteratively since the foundation of software engineering in the 1960s. Guéhéneuc and Khomh (2019) discusses landmark articles, books, and venues in empirical software engineering that indicate the iterative development of the field. Bird et al. (2015) distinguish four "generations" of analyzing software data, i.e., preliminary work, academic experiments, industrial experiments, and "data science everywhere". In this section, we present five iterations of the development of empirical software engineering from a methodological perspective. We additionally take articles and venues into account, which is needed for a holistic understanding of the field's development. We complement our presentation of the evolution of empirical software engineering by presenting the current situation and an outlook in Sect. 4

and the available books on empirical software engineering in chronological order in Sect. 5 on recommended further reading.

3.1 First Iteration: Mid 1960s to Mid 1970s

In the early years of software engineering, empirical studies were rare, and the only research model commonly in use was the analytical method, where different formal theories were advocated devoid of any empirical evaluation (Glass, 1994). According to a systematic literature review of empirical studies performed by Zendler (2001), Grant and Sackman (1967) published the first empirical study in software engineering in 1967. The authors conducted an experiment that compared the performance of two groups of developers, one working with online access to a computer through a terminal and the other with offline access in batch mode. Another empirical study published early in the history of software engineering was an article by Knuth (1971), in which the author studied a set of Fortran programs to understand what developers do in Fortran programs. Akiyama (1971) describes the first known "size law" (Bird et al., 2015), stating that the number of defects is a function of the number of lines of code. The authors in these and other early studies defined the goal of the study, the questions to research, and the measures to answer these questions in an ad-hoc fashion (Guéhéneuc and Khomh, 2019). However, they were pioneers in the application of empirical methods in software engineering.

3.2 Second Iteration: Mid 1970s to Mid 1980s

In the second iteration, already more empirical studies, mainly in-vitro experiments, were conducted. Prominent examples are experiments on structured programming (Lucas et al., 1976), flowcharting (Shneiderman et al., 1977), and software testing (Myers, 1978). The second iteration is characterized by first attempts to provide a systematic methodology to define empirical studies in software engineering in general and experiments in particular. These attempts culminated in the definition of the Goal/Question/Metrics (GQM) approach by Basili and Weiss (1984). The GQM approach helped practitioners and researchers to define measurement programs based on goals related to products, processes, and resources that can be achieved by answering questions that characterize the objects of measurement using metrics. The methodology has been used to define experiments in software engineering systematically.

In that iteration, empirical software engineering was also institutionalized for the first time. In 1976, the NASA Goddard Software Engineering Laboratory (NASA/SEL) was established at the University of Maryland, College Park (USA), aiming at support the observation and understanding of software projects (Basili and Zelkowitz, 2007). The establishment of NASA/SEL provided the means to strengthen

the importance of using basic scientific and engineering concepts in the context of software engineering (McGarry et al., 1994). The paradigm change provided by using GQM (Basili and Weiss, 1984), including the ability of packaging knowledge on how to better build a software system, improved the way experiences could be organized and shared. The building and evolution of models at NASA/SEL pave the road for organizing the Experience Factory model (Basili et al., 1994) and the dissemination of initial good practices on empirical software engineering.

3.3 Third Iteration: Mid 1980s to End of the 1990s

In the third iteration, not only experiments but also surveys (for instance, by Burkhard and Jenster (1989) on the application of computer-aided software engineering tools) and case studies (for instance, by Curtis et al. (1988) on the software design process for large systems) were performed to some extent. Also, the explicit discussion of threats to validity appeared in that iteration. One of the first studies explicitly discussing its threats to validity was an article by Swanson and Beath (1988) on the use of case study data in software management research. From the late 1980s, researchers also started to analyze software data using algorithms taken from artificial intelligence research (Bird et al., 2015). For instance, decision trees and neural networks were applied to predict error-proneness (Porter and Selby, 1990), to estimate software effort (Srinivasan and Fisher, 1995) and to model reliability growth (Tian, 1995).

In the third iteration, empirical studies began to attract the attention of several research groups all over the world, who realized the importance of providing empirical evidence about the developed and investigated software products and processes. The experiences shared by NASA/SEL and the participation of several researchers in conducting experiments together with NASA/SEL helped to strengthen the use of different experimental strategies and the application of surveys.

The interest in the application of the scientific method by different researchers, the identification of the need to evolve the experimentation process through sharing of experimental knowledge among peers as well as the transfer of knowledge to industry, among other reasons, led to the establishment of the International Software Engineering Research Network (ISERN) in 1992. ISERN held its first annual meeting in Japan in 1993 sponsored by the Graduate School of Information Science at the Nara Institute of Science and Technology.

The need to share the ever increasing number of studies and their results, and the growing number of researchers applying empirical methods in software engineering lead to the foundation of suitable forums. In 1993 the IEEE International Software Metrics Symposium, in 1996, the Empirical Software Engineering International Journal, and 1997, the Empirical Assessments in Software Engineering (EASE) event at Keele University were founded.

By the end of this iteration, several institutes dedicated to empirical software engineering were established. In 1996, the Fraunhofer Institute for Experimental Software

Engineering (IESE) associated with the University of Kaiserslautern (Germany) was established. In 1998, the Fraunhofer Center for Experimental Software Engineering (CESE) associated with the University of Maryland, College Park (USA) began operations. Also, other institutions and laboratories, such as National ICT Australia as well as the Simula Research Laboratory and SINTEF (both located in Norway), among others, started to promote empirical studies in software engineering in the industry.

Finally, by the end of the 1990s, the publication of methodological papers on empirical methods in software engineering started. Zelkowitz and Wallace (1998) provided an overview of experimental techniques to validate new technologies, Seaman (1999) provided guidelines for qualitative data collection and analysis, and Basili et al. (1999) discussed families of experiments.

3.4 Fourth Iteration: The 2000s

Since 2000 research methodology has received considerable attention, and therefore the publication of methodological papers further increased. For instance, Höst et al. (2000) discuss the usage of students as subjects in experiments, Shull et al. (2001) describe a methodology to introduce software processes based on experimentation, Pfleeger and Kitchenham (2001) provide guidelines on surveys in software engineering, Lethbridge et al. (2005) provide a classification of data collection methods, Kitchenham and Charters (2007) provide guidelines for performing systematic literature reviews in software engineering, Shull et al. (2008) discuss the role of replication in empirical software engineering, and Runeson and Höst (2009) provide guidelines for case study research. In connection to the increased interest in research methodology, also the first books on empirical research methods in software engineering with a focus on experimentation written by Wohlin et al. (2000) and Juristo and Moreno (2001) appeared around 2000 (see Sect. 5 for a comprehensive overview of books on empirical software engineering). Also, combining research methods and performing multi-method research became more popular in the period. One of the first papers following a multi-method research methodology was published by Espinosa et al. (2002) on shared mental models, familiarity, and coordination in distributed software teams.

With the growing number of empirical studies, knowledge aggregation based on these primary studies became more crucial to understand software engineering phenomena better. No single empirical study on a software engineering phenomenon can be considered definitive (Shull et al., 2004) and generalized to any context. Therefore, the replication of studies in different contexts is of paramount importance to strengthen its findings. However, the existence of conclusive, no conclusive, contradictory, and confirmatory results about a particular software engineering phenomenon should be combined to strengthen the evidence on the software phenomena or to reveal the need for more primary studies on the phenomenon of interest. In consequence, there arose a need for secondary studies that aim to organize, aggre-

gate, and synthesize all relevant results from primary studies regarding a particular phenomenon under research. Kitchenham (2004) was the first who recommended the use of systematic literature reviews (SLRs) in software engineering and adapted respective guidelines, mainly from medical research, to software engineering. With the guidelines of Kitchenham (2004) and Biolchini (2005), the empirical software engineering community had a tool to systematically synthesize knowledge available in primary studies, which spread rapidly and enabled evidence-based software engineering (Kitchenham et al., 2004). In a systematic review of SLRs in software engineering, Kitchenham and Brereton (2013) identified 68 papers reporting 63 unique SLRs published in SE conferences and journals between 2005 and mid-2012. Petersen et al. (2008) clarify and expand upon the differences between SLRs and with the growing number of empirical studies, knowledge aggregation based on these primary studies became more crucial to understand software engineering phenomena better. No single empirical study on a software engineering phenomenon can be considered definitive (Shull et al., 2004) and generalized to any context. Therefore, the replication of studies in different contexts is of paramount importance to strengthen its findings. However, the existence of conclusive, no conclusive, contradictory, and confirmatory results about a particular software engineering phenomenon should be combined to strengthen the evidence on the software phenomena or to reveal the need for more primary studies on the phenomenon of interest. In consequence, there arose a need for secondary studies that aim to organize, aggregate, and synthesize all relevant results from primary studies regarding a particular phenomenon under research. Kitchenham (2004) was the first who recommended the use of systematic literature reviews (SLRs) in software engineering and adapted respective guidelines, mainly from medical research, to software engineering. With the guidelines of Kitchenham (2004) and Biolchini (2005), the empirical software engineering community had a tool to systematically synthesize knowledge available in primary studies, which spread rapidly and enabled evidence-based software engineering (Kitchenham et al., 2004). In a systematic review of SLRs in software engineering, Kitchenham and Brereton (2013) identified 68 papers reporting 63 unique SLRs published in SE conferences and journals between 2005 and mid-2012. Petersen et al. (2008) clarify and expand upon the differences between SLRs and systematic mapping studies, and provide guidelines for the latter. In their seminal paper on the future of empirical methods in software engineering research, Sjoberg et al. (2007) present the important role of synthesis of empirical evidence in their vision of software engineering research. The vision is that for all fields of software engineering, empirical research methods should enable the development of scientific knowledge about how useful different SE technologies are for different kinds of actors, performing different kinds of activities, on different kinds of systems to guide the development of new SE technology and important SE decisions in industry. Major challenges to the pursuit of this vision are more and better synthesis of empirical evidence, and connected to that building and testing more theories as well as increasing quality, including relevance, of studies.

One of the problems faced by the software engineering community has often been the scarcity of software data for conducting empirical studies (Malhotra, 2016). The

availability of (open) source code repositories and software process data due to automated or even continuous software engineering enabled new data mining approaches in software engineering in that period. In a seminal paper, Zimmermann et al. (2005) used association rule learning to find patterns of defects in a large set of open-source projects. Furthermore, also, software data from companies were analyzed. For instance, at AT&T, Ostrand et al. (2004) code metrics to predict defects, and at Microsoft – which even founded an own Empirical Software Engineering (ESE) group in Microsoft Research (Bird et al., 2011) – Nagappan and Ball (2005) showed that data from that organization could predict software quality. In consequence, also repositories – like the PROMISE repository – that collect software data and make them publicly available were founded. The PROMISE repository was founded in 2005 and seeded with NASA data (Menzies et al., 2014). The empirical evidence gathered through analyzing the data collected from the software repositories is considered to be an important support for the (empirical) software engineering community these days. There are even venues that focus on analysis of software data such as Mining Software Repositories (MSR), which was organized for the first time in 2004 in Edinburgh (UK) and Predictive Models and Data Analytics in Software Engineering (PROMISE), which was organized for the first in 2005 in St. Louis (USA).

In general, the growing interest in empirical software engineering in that period resulted in projects such as the Experimental Software Engineering Research Network (ESERNET) in Europe from 2001 to 2003 and the foundation of several venues. In 2007, the first ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM) was held in Madrid (Spain). ESEM is the result of the merger between the ACM/IEEE International Symposium on Empirical Software Engineering, which ran from 2002 to 2006, and the IEEE International Software Metrics Symposium, which ran from 1993 to 2005. In 2003, Experimental Software Engineering Latin American Workshop (ESELAW) was organized for the first time. Also, in 2003, the International Advanced School on Empirical Software Engineering (IASESE) performed its first set of classes in Rome (Italy). In 2006, the International Doctoral Symposium on Empirical Software Engineering (IDoESE) was founded. Today, the ISERN annual meeting, IASESE, IDoESE, and ESEM form the Empirical Software Engineering International Week (ESEIW), which is held annually.

3.5 Fifth iteration: The 2010s

Since 2010 empirical studies are "everywhere" in software engineering. Almost all papers in major software engineering conferences like ICSE contain empirical studies. Also, more and more books dedicated to empirical research methodology in software engineering are published (see Sect. 5), and papers on empirical research methodology are published at a constant pace. For instance, Ivarsson and Gorschek (2011) present a model for evaluating the rigor and relevance of technology evalu-

ations in industry, Arcuri and Briand (2014) provide a guide to statistical tests for assessing randomized algorithms in software engineering, Wieringa (2014a) discusses scaling up of empirical methods for technology validation in practice, Wohlin and Aurum (2015) provide a decision-making structure for selecting a research design, de Mello et al. (2015) provide probabilistic sampling approaches for large-scale surveys, Sharp et al. (2016) discuss the use and value of ethnographic studies in software engineering research, Stol et al. (2016) discuss the use of grounded theory and their reporting, Briand et al. (2017) discuss the importance of context and the overrating of generalizability in software engineering, and Stol and Fitzgerald (2018) provide a holistic framework for software engineering research. Furthermore, Harman et al. (2010) provide a comprehensive overview and guidance on the application of search-based optimization in software engineering. Especially, in this period many papers presenting results on search-based software engineering, that generally (though not exclusively) fall in the category of empirical software engineering papers were published. Due to the potentially high computational complexity of optimization algorithms, some researchers have started to use high performance computing environments to support the execution of their studies (Farzat et al., 2019).

In this iteration, one can observe a growing interest in the role of theory within software engineering research to develop the field further as a scientific discipline. In December 2009, the Software Engineering Method and Theory (SEMAT) initiative was launched that aims towards the development of a general theory of software engineering. SEMAT organized several events, among others, a workshop series on a General Theory of Software Engineering (GTSE) between 2012 and 2015. Stol and Fitzgerald (2015) even argue for a theory-oriented software engineering research perspective, which can complement the recent focus on evidence-based software engineering. Also, several concrete theories have been developed in that iteration. For instance, Johnson and Ekstedt (2016) present a general theory of software engineering called Tarpit, Bjarnason et al. (2016) a theory of distances in software engineering, and Wagner et al. (2019) a theory on requirements engineering.

Today not only almost all papers in major software engineering conferences contain empirical studies, but also most software engineering conferences have explicitly integrated empirical software engineering into their program, e.g., as dedicated sessions or tracks. In addition, there are several workshops on conducting empirical studies in specific areas. For instance, at ICSE, there has been a collocated International Workshop on Conducting Empirical Studies in Industry (CESI) and at RE the International Workshop on Empirical Requirements Engineering (EmpiRE). The Experimental Software Engineering Latin American Workshop (ESELAW) joined the Ibero-American Conference on Software Engineering (CIbSE) in 2011 as a co-located workshop and became a dedicated track in 2013 due to the increased number of submissions.

ESEIW, including ESEM and EASE are established as the leading annual events to discuss methodological issues on empirical research in software engineering. Empirical methods have been an explicit topic in several summer schools including the annual LASER summer school (which hosted the topic empirical software engineering in 2010), PASED - Canadian Summer School on Practical Analyses of

Software Engineering Data in 2011, the Empirical Research Methods in Software Engineering and Informatics (ERMSEI) in 2016 and 2017, the International Summer School on Software Engineering (SIESTA) in 2018 and 2019 as well as the 2019 Summer School in Empirical Software Engineering at Brunel (UK). In the context of ESEIW, the International Advanced School on Empirical Software Engineering (IASESE) has been organized annually since 2003 and helped to spread knowledge on current empirical methods in software engineering among junior and senior researchers. Figure 1 presents the IASESE timeline and its topics along the places and years. The topics taught over the years also reflect the evolution of empirical software engineering, as discussed in this section.
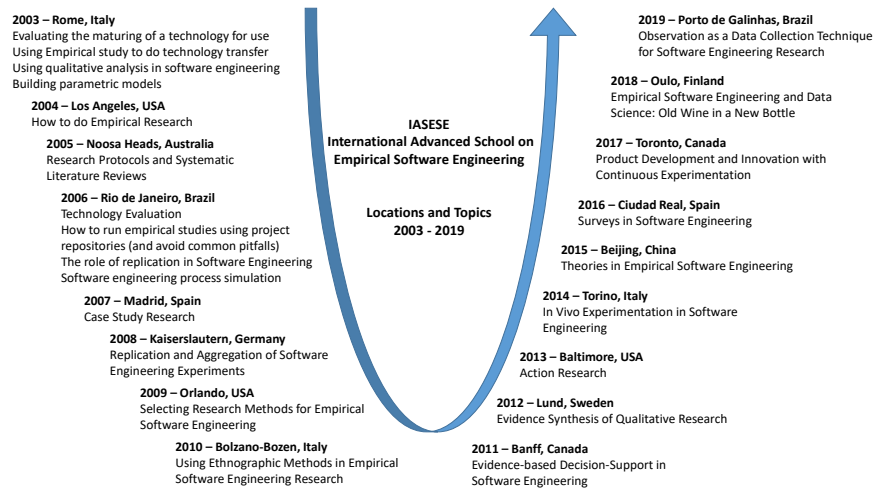
2003 – Rome, Italy
Evaluating the maturing of a technology for use
Using Empirical study to do technology transfer
Using qualitative analysis in software engineering
Building parametric models

2004 – Los Angeles, USA
How to do Empirical Research

2005 – Noosa Heads, Australia
Research Protocols and Systematic
Literature Reviews

2006 – Rio de Janeiro, Brazil
Technology Evaluation
How to run empirical studies using project
repositories (and avoid common pitfalls)
The role of replication in Software Engineering
Software engineering process simulation

2007 – Madrid, Spain
Case Study Research

2008 – Kaiserslautern, Germany
Replication and Aggregation of Software
Engineering Experiments

2009 – Orlando, USA
Selecting Research Methods for Empirical
Software Engineering

2010 – Bolzano-Bozen, Italy
Using Ethnographic Methods in Empirical
Software Engineering Research

IASESE
International Advanced School on
Empirical Software Engineering

Locations and Topics
2003 - 2019

2019 – Porto de Galinhas, Brazil
Observation as a Data Collection Technique
for Software Engineering Research

2018 – Oulo, Finland
Empirical Software Engineering and Data
Science: Old Wine in a New Bottle

2017 – Toronto, Canada
Product Development and Innovation with
Continuous Experimentation

2016 – Ciudad Real, Spain
Surveys in Software Engineering

2015 – Beijing, China
Theories in Empirical Software Engineering

2014 – Torino, Italy
In Vivo Experimentation in Software
Engineering

2013 – Baltimore, USA
Action Research

2012 – Lund, Sweden
Evidence Synthesis of Qualitative Research

2011 – Banff, Canada
Evidence-based Decision-Support in
Software Engineering

Fig. 1 International Advanced School on Empirical Software Engineering (IASESE) Timeline and Topics from 2003 to 2019

4 Current Situation and Outlook

Since the first empirical studies in the 1960s, the field of empirical software engineering has considerably matured in several iterations. However, the empirical methods resulting from the five iterations presented in the previous section are not the end of the story, and as in any scientific discipline, research methods develop further. The chapters of this book discuss contemporary empirical methods that impact the current evolution of empirical software engineering and form the backbone of its next iteration. For sure, the description of the current situation and future trends is never complete and always subjective to some extent. But we think that the chap-

ters covered in this book show several interesting trends in contemporary empirical methods in software engineering, which we want to summarize here.

The evolution of empirical software engineering leads to the continuous adoption of empirical methods from other fields and the refinement of existing empirical methods in software engineering. The resulting plurality of research methods requires guidance in knowledge-seeking and solution-seeking (i.e., design science) research. Chap. 2 presents guidelines for conducting software engineering research based on the *ABC framework*, where ABC represents the three desirable aspects of research generalizability over actors (A), precise control of behavior (B), and realism of context (C). Each empirical method has its strengths and weaknesses. It is beneficial to utilize a mix of methods depending on the research goal or even to combine methods. Case survey research combines case study and survey research, which rely primarily on qualitative and quantitative data, respectively. Chap. 3 provides an overview of the *case survey method*. While being an important and often used empirical method, survey research has been less discussed on a methodological level than other types of empirical methods. Chap. 4 discusses *methodological issues in survey research* for software engineering concerning theory building, sampling, invitation and follow-up, statistical analysis, qualitative analysis, and assessment of psychological constructs. Although software engineering is an engineering discipline, the design science paradigm has been explicitly adapted to software engineering relatively late by Wieringa (2014b), and the full potential of the design science paradigm has not been exploited so far in software engineering. Chap. 5 uses the *design science paradigm* as a frame for empirical software engineering and uses it to support the assessment of research contributions, industry-academia communication, and theoretical knowledge building.

It is generally acknowledged that software development is a human-intensive activity as software is built by humans for humans. However, traditionally SE research has focused on artifacts and processes without explicitly taking human factors in general and the developer perspective in particular into account. If the perspective on how developers work was considered, then it was mostly measured from a subjective perspective, e.g., by interviews or opinion surveys, or a "black-box" perspective by mining repository data or measuring the created development artifacts. Chap. 6 introduces *biometric sensors and measure* that provide new opportunities to more objectively measure physiological changes in the human body that can be linked to various psychological processes. These biometric measurements can be used to gain insights on fundamental cognitive and emotional processes of software developers while they are working, but also to provide better and more prompt tool support for developers. Another human-related issue is the involvement of humans in empirical studies, especially in experiments. On the one hand, it normally difficult to recruit a significant number of professionals for an empirical study, and on the other hand, measurements are invasive. Chap. 7 explores the potential of *human computation methods*, such as crowdsourcing, for experimentation in empirical software engineering.

Empirical methods rely on the collected data. However, the volume, velocity, and variety of data in software products and processes have exploded during the

last years. Therefore, the new scientific paradigm of *data science* has gained much attention, also within software engineering. Chap. 8 relates to traditional ESE and data science. It shows that both paradigms have many characteristics in common and can benefit from each other. Given large data sets, *optimization* is an important form of data analytics support of human decision-making. Empirical studies serve both as a model and as data input for optimization. Chap. 9 provides an overview of optimization in software engineering in general and its value and applicability in ESE in particular. With increased automation, uncertainty (due to the application of statistical models), and monitoring capabilities in data-driven software engineering, also the role of *simulation* techniques becomes more important. Chap. 10 provides a guide to simulation-based studies in software engineering. *Bayesian data analysis* is a means to embrace uncertainty by using multilevel statistical models and making use of all available information at hand. Chap. 11 provides an introduction to Bayesian data analysis and an application example to empirical software engineering dealing with common issues in ESE like missing data.

Extracting, aggregating, and synthesizing evidence from empirical studies is essential for the development of scientific knowledge and the field of software engineering. However, conducting secondary studies like systematic literature reviews and aggregating evidence is still challenging. Conducting systematic literature reviews (SLRs) is largely a manual and, therefore, time-consuming and error-prone process. Chap. 12 provides strategies to *automate the SLR process*. Secondary studies often lack connection to software engineering practice, which is essential to software engineering. Chap. 13 presents the concept of *rapid reviews*, which are lightweight secondary studies focused on delivering evidence to practitioners on time. Another approach to link to practice is to take grey literature into account in empirical studies. Chap. 14 discusses the concept of *grey literature* in software engineering and ways how to consider it in primary and secondary studies. Considering that secondary studies are often used to support the evidence-based paradigm, it is crucial to managing their threats properly. Chap. 15 provides guidelines for managing *threats to validity of secondary studies* in software engineering. Evidence in software engineering is often rare and produced in both quantitative and qualitative forms. It makes the synthesis of evidence, which is an essential element in scientific knowledge creation, a challenging task. Chap. 16 provides an overview of research *synthesis methods* in software engineering.

Society in general and funding agencies, in particular, put a stronger focus on the impact of (software engineering) research. Therefore, open science and research transfer are becoming essential topics in (empirical) software engineering. *Open science* describes the movement of making any research artifact available to the public and includes open access, open data, and open source. The topic is natural and especially important in empirical software engineering to guarantee the replicability of empirical studies. Chap. 17 reflects upon the essentials in open science for software engineering to help to establish a common ground and to make open science a norm in SE. Industry-academia collaboration is one of the cornerstones of empirical software engineering. However, close and sustainable collaboration with industry

are key issues in the field. Chap. 18 presents a seven-step *industrial co-production* approach that enables deep and long-term industry-academia collaboration.

5 Recommended Further Reading

Since 2000 research methodology has received considerable attention in the software engineering research community. Therefore, plenty of literature is available on empirical research methodology in software engineering. Molléri et al. (2019) identified in a recent systematic mapping study 341 methodological papers on empirical research in software engineering - and therefore, a complete overview would exceed the scope of this book chapter. However, following the style of this book chapter, we provide an overview of the available English text and special issue books explicitly dedicated to empirical research methodology in software engineering in chronological order of their publication.

Wohlin et al. (2000) published a book entitled "Experimentation in Software Engineering", which provides an overview of the core empirical strategies in software engineering, i.e., surveys, experimentation and case studies and as its main content all steps in the experimentation process, i.e., scoping, planning, operation, analysis and interpretation as well as presentation and package. The book is complemented by exercises and examples, e.g., an experiment comparing different programming languages. Consequently, the book targets students, teachers, researchers, and practitioners in software engineering. In 2012 a revision of this popular book had been published with Springer (Wohlin et al., 2012).

Juristo and Moreno (2001) published a book entitled "Basics of Software Engineering Experimentation", which presents the basics of designing and analyzing experiments both to software engineering researchers and practitioners based on SE examples like comparing the effectiveness of defect detection techniques. The book presents the underlying statistical methods, including the computation of test statistics in detail.

Endres and Rombach (2003) published "A handbook of software and systems engineering. Empirical observations, laws, and theories". The book presents rules, laws, and their underlying theories from all phases of the software development lifecycle. The book provides the reader with clear statements of software and system engineering laws and their applicability as well as related empirical evidence. The consideration of empirical evidence distinguishes the book from other available handbooks and textbooks on software engineering.

Juristo and Moreno (2003) edited "Lecture notes on empirical software engineering", which aims to spread the idea of the importance of empirical knowledge in software development from a highly practical viewpoint. It defines the body of empirically validated knowledge in software development to advise practitioners on what methods or techniques have been empirically analyzed and what the results were. Furthermore, it promotes "empirical tests", which have traditionally been car-

ried out by universities or research centers, for application in industry to validate software development technologies used in practice.

Shull et al. (2007) published the "Guide to advanced empirical software engineering". It is an edited book written by experts in empirical software engineering. It covers advanced research methods and techniques, practical foundations, as well as knowledge creation, approaches. The book at hand provides a continuation of that seminal book covering recent developments in empirical software engineering.

Runeson et al. (2012) published a book entitled "Case study research in software engineering: Guidelines and examples", which covers guidelines for all steps of case study research, i.e., design, data collection, data analysis and interpretation, as well as reporting and dissemination. The book is complemented with examples from extreme programming, project management, quality monitoring as well as requirements engineering and additionally also provides checklists.

Wieringa (2014b) published a book entitled "Design science methodology for information systems and software engineering", which provides guidelines for practicing design science in software engineering research. A design process usually iterates over two activities, i.e., first designing an artifact that improves something for stakeholders, and subsequently empirically validating the performance of that artifact in its context. This "validation in context" is a key feature of the book.

Menzies et al. (2014) published a book entitled "Sharing data and models in software engineering". The central theme of the book is how to share what has been learned by data science from software projects. The book is driven by the PROMISE (Predictive Models and Data Analytics in Software Engineering) community. It is the first book dedicated to data science in software and mining software repositories. Closely related to this book, Bird et al. (2015) published a book entitled "The art and science of analyzing software data", which is driven by the MSR (Mining Software Repositories) community and focuses mainly on data analysis based on statistics and machine learning. Another related book published by Menzies et al. (2016) covers perspectives on data science for software engineering by various authors.

Kitchenham et al. (2015) published a book entitled "Evidence-based software engineering and systematic reviews", which provides practical guidance on how to conduct secondary studies in software engineering. The book also discusses the nature of evidence and explains the types of primary studies that provide inputs to a secondary study.

Malhotra (2016) published a book entitled "Empirical research in software engineering: concepts, analysis, and applications", which shows how to implement empirical research processes, procedures, and practices in software engineering. The book covers many accompanying exercises and examples. The author especially also discusses the process of developing predictive models, such as defect prediction and change prediction, on data collected from source code repositories, and, more generally the application of machine learning techniques in empirical software engineering.

ben Othmane et al. (2017) published a book entitled "Empirical Research for Software Security: Foundations and Experience", which discusses empirical methods with a special focus on software security.

Staron (2019) published a book entitled "Action Research in Software Engineering: Theory and Applications" , which offers a comprehensive discussion on the use of action research as an instrument to evolve software technologies and promote synergy between researchers and practitioners.

In addition to these textbooks, there are also edited books available that are related to special events in empirical software engineering and cover valuable methodological contributions.

Rombach et al. (1993) edited proceedings from a Dagstuhl seminar in 1992 on empirical software engineering entitled "Experimental Software Engineering Issues: Critical Assessment and Future Directions". The goal was to discuss the state of the art of empirical software engineering by assessing past accomplishments, raising open questions, and proposing a future research agenda at that time. However, many contributions of that book are still relevant today.

Conradi and Wang (2003) edited a book entitled "Empirical methods and studies in software engineering: experiences from ESERNET", which covers experiences from the Experimental Software Engineering Research NETwork (ESERNET), a thematic network funded by the European Union between 2001 and 2003.

Boehm et al. (2005) edited a book entitled "Foundations of empirical software engineering: the legacy of Victor R. Basili" on the occasion of V. R. Basili's 65th birthday, which covers reprints of 20 papers that defined much of his work.

Basili et al. (2007) edited proceedings from another Dagstuhl seminar in 2006 on empirical software engineering entitled "Empirical Software Engineering Issues. Critical Assessment and Future Directions".

Münch and Schmid (2013) edited a book entitled "Perspectives on the future of software engineering: essays in honor of Dieter Rombach" on the occasion of Dieter Rombach's 60th birthday, which covers contributions by renown researchers and colleagues of him.

6 Conclusion

In this chapter we presented the evolution of empirical software engineering in five iterations, i.e., (1) mid-1960s to mid-1970s, (2) mid-1970s to mid-1980s, (3) mid-1980s to end of the 1990s, (4) the 2000s, and (5) the 2010s. We presented the five iterations of the development of empirical software engineering mainly from a methodological perspective and additionally took key papers, venues, and books into account. Available books explicitly dedicated to empirical research methodology in software engineering were covered in chronological order in a separate section on recommended further readings. Furthermore, we discuss – based on the chapters in this book – trends on contemporary empirical methods in software engineering related to the plurality of research methods, human factors, data collection and processing, aggregation and synthesis of evidence, and impact of software engineering research.

References

Akiyama F (1971) An example of software system debugging. In: IFIP Congress (1), vol 71, pp 353–359

Araújo MAP, Monteiro VF, Travassos GH (2012) Towards a model to support studies of software evolution. In: Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement - ESEM '12, ACM Press, pp 281–290, DOI 10.1145/2372251. 2372303

Arcuri A, Briand L (2014) A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering. Software Testing, Verification and Reliability 24(3):219–250

Baros MO, Werner CML, Travassos GH (2004) Supporting risks in software project management. Journal of Systems and Software 70(1):21 – 35, DOI 10.1016/S0164-1212(02)00155-3

Basili V, Rombach D, Schneider K, Kitchenham B, Pfahl D, Selby R (2007) Empirical Software Engineering Issues. Critical Assessment and Future Directions: International Workshop, Dagstuhl Castle, Germany, June 26-30, 2006, Revised Papers, vol 4336. Springer

Basili VR (1993) The experimental paradigm in software engineering. In: Experimental Software Engineering Issues: Critical Assessment and Future Directions, Springer, pp 1–12

Basili VR, Weiss DM (1984) A methodology for collecting valid software engineering data. IEEE Transactions on Software Engineering SE-10(6):728–738, DOI 10.1109/TSE.1984.5010301

Basili VR, Zelkowitz MV (2007) Empirical studies to build a science of computer science. Commun ACM 50(11):33–37

Basili VR, Caldiera G, Rombach HD (1994) Experience factory. Encyclopedia of Software Engineering 1:469–476

Basili VR, Shull F, Lanubile F (1999) Building knowledge through families of experiments. IEEE Transactions on Software Engineering 25(4):456–473

Bertin E (1978) Qualitative and Semiquantitative Analysis, Springer

Biolchini MPNATG J (2005) Systematic review in software engineering: Relevance and utility. Tech. rep.

Bird C, Murphy B, Nagappan N, Zimmermann T (2011) Empirical software engineering at microsoft research. In: Proceedings of the ACM 2011 conference on Computer supported cooperative work, ACM, pp 143–150

Bird C, Menzies T, Zimmermann T (2015) The art and science of analyzing software data. Elsevier

Bjarnason E, Smolander K, Engström E, Runeson P (2016) A theory of distances in software engineering. Information and Software Technology 70:204–219

Boehm B, Rombach HD, Zelkowitz MV (2005) Foundations of empirical software engineering: the legacy of Victor R. Basili. Springer Science & Business Media

Briand L, Bianculli D, Nejati S, Pastore F, Sabetzadeh M (2017) The case for context-driven software engineering research: Generalizability is overrated. IEEE Software 34(5):72–75

Burkhard DL, Jenster PV (1989) Applications of computer-aided software engineering tools: survey of current and prospective users. ACM SIGMIS Database: the DATABASE for Advances in Information Systems 20(3):28–37

Conradi R, Wang AI (2003) Empirical methods and studies in software engineering: experiences from ESERNET, vol 2765. Springer

Creswell JW, Creswell JD (2018) Research design: Qualitative, quantitative, and mixed methods approaches. Sage publications

Curtis B, Krasner H, Iscoe N (1988) A field study of the software design process for large systems. Communications of the ACM 31(11):1268–1287

Endres A, Rombach HD (2003) A handbook of software and systems engineering: Empirical observations, laws, and theories. Pearson Education

Espinosa A, Kraut R, Slaughter S, Lerch J, Herbsleb J, Mockus A (2002) Shared mental models, familiarity, and coordination: A multi-method study of distributed software teams. ICIS 2002 Proceedings p 39

Farzat F, Barros MO, Travassos GH (2019) Evolving javascript code to reduce load time. Ttransactions on Software Engineering DOI 10.1109/TSE.2019.2928293

Fink A (2003) The survey handbook. Sage

Glass RL (1994) The software-research crisis. IEEE Software 11(6):42–47

Grant EE, Sackman H (1967) An exploratory investigation of programmer performance under on-line and off-line conditions. IEEE Transactions on Human Factors in Electronics (1):33–48

Guéhéneuc YG, Khomh F (2019) Empirical software engineering. In: Cha S, Taylor RN, Kang KC (eds) Handbook of Software Engineering, Springer, pp 285–320

Harman M, McMinn P, De Souza JT, Yoo S (2010) Search based software engineering: Techniques, taxonomy, tutorial. In: Empirical software engineering and verification, Springer, pp 1–59

Harrison W, Basili VR (1996) Editorial. Empirical Software engineering 1:5–10, DOI https://doi.org/10.1007/BF00125808

Hey AJ, Hey T, Pápay G (2014) The computing universe: a journey through a revolution. Cambridge University Press

Höst M, Regnell B, Wohlin C (2000) Using students as subjects - a comparative study of students and professionals in lead-time impact assessment. Empirical Software Engineering 5(3):201–214

IEEE (1990) 610.12-19919 - ieee standard glossary of software engineering terminology

IEEE (2010) Iso/iec/ieee 24765:2010 systems and software engineering - vocabulary

Ivarsson M, Gorschek T (2011) A method for evaluating rigor and industrial relevance of technology evaluations. Empirical Software Engineering 16(3):365–395

Johnson P, Ekstedt M (2016) The tarpit–a general theory of software engineering. Information and Software Technology 70:181–203

Juristo N, Moreno AM (2001) Basics of software engineering experimentation. Springer Science & Business Media

Juristo N, Moreno AM (2003) Lecture notes on empirical software engineering, vol 12. World Scientific

Kitchenham B (2004) Procedures for performing systematic reviews. Tech. rep.

Kitchenham B, Brereton P (2013) A systematic review of systematic review process research in software engineering. Information and software technology 55(12):2049–2075

Kitchenham B, Charters S (2007) Guidelines for performing systematic literature reviews in software engineering. Tech. rep.

Kitchenham BA, Dyba T, Jorgensen M (2004) Evidence-based software engineering. In: Proceedings of the 26th international conference on software engineering, IEEE Computer Society, pp 273–281

Kitchenham BA, Budgen D, Brereton P (2015) Evidence-based software engineering and systematic reviews, vol 4. CRC press

Knuth DE (1971) An empirical study of fortran programs. Software: Practice and experience 1(2):105–133

Lethbridge TC, Sim SE, Singer J (2005) Studying software engineers: Data collection techniques for software field studies. Empirical software engineering 10(3):311–341

Lucas J, Henry C, Kaplan RB (1976) A structured programming experiment. The Computer Journal 19(2):136–138

Lyu MR, et al. (1996) Handbook of software reliability engineering, vol 222. IEEE computer society press CA

Malhotra R (2016) Empirical research in software engineering: concepts, analysis, and applications. Chapman and Hall/CRC

McGarry F, Pajerski R, Page G, Waligora S, Basili V, Zelkowitz M (1994) Software process improvement in the nasa software engineering laboratory. Tech. Rep. CMU/SEI-94-TR-022,

Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, URL `http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=12241`

de Mello RM, Da Silva PC, Travassos GH (2015) Investigating probabilistic sampling approaches for large-scale surveys in software engineering. Journal of Software Engineering Research and Development 3(1):8

Menzies T, Kocaguneli E, Turhan B, Minku L, Peters F (2014) Sharing data and models in software engineering. Morgan Kaufmann

Menzies T, Williams L, Zimmermann T (2016) Perspectives on Data Science for Software Engineering. Morgan Kaufmann

Molléri JS, Petersen K, Mendes E (2019) Cerse-catalog for empirical research in software engineering: A systematic mapping study. Information and Software Technology 105:117–149

Münch J, Schmid K (2013) Perspectives on the future of software engineering: essays in honor of Dieter Rombach. Springer Science & Business Media

Myers GJ (1978) A controlled experiment in program testing and code walkthroughs/inspections. Communications of the ACM 21(9):760–768

Nagappan N, Ball T (2005) Use of relative code churn measures to predict system defect density. In: Proceedings of the 27th international conference on Software engineering, ACM, pp 284–292

Ogborn J, Miller R (1994) Computational Issues in Modelling, The Falmer Press

Ostrand TJ, Weyuker EJ, Bell RM (2004) Where the bugs are. In: ACM SIGSOFT Software Engineering Notes, ACM, vol 29, pp 86–96

ben Othmane L, Jaatun MG, Weippl E (2017) Empirical Research for Software Security: Foundations and Experience. CRC Press

Petersen K, Feldt R, Mujtaba S, Mattsson M (2008) Systematic mapping studies in software engineering. In: Ease, vol 8, pp 68–77

Petersen K, Vakkalanka S, Kuzniarz L (2015) Guidelines for conducting systematic mapping studies in software engineering: An update. Information and Software Technology 64:1–18

Pfleeger SL (1995) Experimental design and analysis in software engineering. Annals of Software Engineering 1(1):219–253

Pfleeger SL, Kitchenham BA (2001) Principles of survey research: part 1: turning lemons into lemonade. ACM SIGSOFT Software Engineering Notes 26(6):16–18

Porter AA, Selby RW (1990) Empirically guided software development using metric-based classification trees. IEEE software 7(2):46–54

Rombach HD, Basili VR, Selby RW (1993) Experimental Software Engineering Issues:: Critical Assessment and Future Directions. International Workshop, Dagstuhl Castle, Germany, September 14-18, 1992. Proceedings, vol 706. Springer Science & Business Media

Runeson P, Höst M (2009) Guidelines for conducting and reporting case study research in software engineering. Empirical software engineering 14(2):131

Runeson P, Höst M, Rainer A, Regnell B (2012) Case study research in software engineering. In: Guidelines and examples, Wiley Online Library

Seaman CB (1999) Qualitative methods in empirical studies of software engineering. IEEE Transactions on software engineering 25(4):557–572

Shadish WR, Cook TD, Campbell DT (2002) Experimental and quasi-experimental designs for generalized causal inference

Sharp H, Dittrich Y, De Souza CR (2016) The role of ethnographic studies in empirical software engineering. IEEE Transactions on Software Engineering 42(8):786–804

Shneiderman B, Mayer R, McKay D, Heller P (1977) Experimental investigations of the utility of detailed flowcharts in programming. Communications of the ACM 20(6):373–381

Shull F, Carver J, Travassos GH (2001) An empirical methodology for introducing software processes. In: ACM SIGSOFT Software Engineering Notes, ACM, vol 26, pp 288–296

Shull F, Mendoncça MG, Basili V, Carver J, Maldonado JC, Fabbri S, Travassos GH, Ferreira MC (2004) Knowledge-sharing issues in experimental software engineering. Empirical Software Engineering 9(1-2):111–137

Shull F, Singer J, Sjøberg DI (2007) Guide to advanced empirical software engineering. Springer

Shull FJ, Carver JC, Vegas S, Juristo N (2008) The role of replications in empirical software engineering. Empirical software engineering 13(2):211–218

Sjoberg DI, Dyba T, Jorgensen M (2007) The future of empirical methods in software engineering research. In: 2007 Future of Software Engineering, IEEE Computer Society, pp 358–378

Srinivasan K, Fisher D (1995) Machine learning approaches to estimating software development effort. IEEE Transactions on Software Engineering 21(2):126–137

Staron M (2019) Action Research in Software Engineering: Theory and Applications. Springer

Stol KJ, Fitzgerald B (2015) Theory-oriented software engineering. Science of Computer Programming 101:79–98

Stol KJ, Fitzgerald B (2018) The abc of software engineering research. ACM Transactions on Software Engineering and Methodology (TOSEM) 27(3):11

Stol KJ, Ralph P, Fitzgerald B (2016) Grounded theory in software engineering research: a critical review and guidelines. In: 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), IEEE, pp 120–131

Swanson EB, Beath CM (1988) The use of case study data in software management research. Journal of Systems and Software 8(1):63–71

Theisen C, Dunaiski M, Williams L, Visser W (2017) Writing good software engineering research papers: revisited. In: Proceedings of the 39th International Conference on Software Engineering Companion, IEEE Press, pp 402–402

Thomke SH (2003) Experimentation matters: unlocking the potential of new technologies for innovation. Harvard Business Press

Tian J (1995) Integrating time domain and input domain analyses of software reliability using tree-based models. IEEE Transactions on Software Engineering 21(12):945–958

Travassos GH, Barros MO (2003) Contributions of in virtuo and in silico experiments for the future of empirical studies in software engineering. In: 2nd Workshop on Empirical Software Engineering the Future of Empirical Studies in Software Engineering, pp 117–130

Travassos GH, dos Santos PSM, Mian PG, Neto ACD, Biolchini J (2008) An environment to support large scale experimentation in software engineering. In: 13th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2008), IEEE, pp 193–202

Wagner S, Fernández DM, Felderer M, Vetrò A, Kalinowski M, Wieringa R, Pfahl D, Conte T, Christiansson MT, Greer D, et al. (2019) Status quo in requirements engineering: A theory and a global family of surveys. ACM Transactions on Software Engineering and Methodology (TOSEM) 28(2):9

Widman L (1989) Expert system reasoning about dynamic systems by semi-quantitative simulation. Computer Methods and Program in Biomedicine - Artificial Intelligence in Medicine 6(3):229–247

Wieringa R (2014a) Empirical research methods for technology validation: Scaling up to practice. Journal of systems and software 95:19–31

Wieringa RJ (2014b) Design science methodology for information systems and software engineering. Springer

Wohlin C, Aurum A (2015) Towards a decision-making structure for selecting a research design in empirical software engineering. Empirical Software Engineering 20(6):1427–1455

Wohlin C, Runeson P, Höst M, Ohlsson M, Regnell B, Wesslén A (2000) Introduction to experimentation in software engineering

Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A (2012) Experimentation in software engineering. Springer Science & Business Media

Zelkowitz MV, Wallace DR (1998) Experimental models for validating technology. Computer 31(5):23–31

Zendler A (2001) A preliminary software engineering theory as investigated by published experiments. Empirical Software Engineering 6(2):161–180

Zimmermann T, Zeller A, Weissgerber P, Diehl S (2005) Mining version histories to guide software changes. IEEE Transactions on Software Engineering 31(6):429–445