CORA Cork Open Research Archive
Cartlann Taighde Oscailte Chorcaí

Title	Guidelines for conducting software engineering research
Authors	Stol, Klaas-Jan;Fitzgerald, Brian
Publication date	2020-08-01
Original Citation	Stol, K.-J. and Fitzgerald, B. (2020) 'Guidelines for Conducting Software Engineering Research', in Felderer, M. & Travassos, G.H. (eds.) Contemporary Empirical Methods in Software Engineering. Cham: Springer International Publishing, pp. 27-62. doi: 10.1007/978-3-030-32489-6_2
Type of publication	Book chapter
Link to publisher's version	https://link.springer.com/chapter/10.1007/978-3-030-32489-6_2 - 10.1007/978-3-030-32489-6_2
Rights	© Springer Nature Switzerland AG 2020. This is a post-peer-review, pre-copyedit version. The final authenticated version is available online at: http://dx.doi.org/10.1007/978-3-030-32489-6_2
Download date	2026-11-03 17:52:12
Item downloaded from	https://hdl.handle.net/10468/10565



UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Guidelines for Conducting Software Engineering Research

Klaas-Jan Stol and Brian Fitzgerald

Abstract This chapter presents a holistic overview of software engineering research strategies. It identifies the two main modes of research within the software engineering research field, namely knowledge-seeking and solution-seeking research—the Design Science model corresponding well with the latter. We present the ABC framework for research strategies as a model to structure knowledge-seeking research. The ABC represents three desirable aspects of research—generalizability over actors (A), precise control of behavior (B) and realism of context (C). Unfortunately, as our framework illustrates, these three aspects cannot be simultaneously maximised. We describe the two dimensions that provide the foundation of the ABC framework—generalizability and control, explain the four different types of settings in which software engineering research is conducted, and position eight archetypal research strategies within the ABC framework. We illustrate each strategy with examples, identify appropriate metaphors, and present an example of how the ABC framework can be used to design a research programme.

1 Introduction

Research methodology—the study of research methods—is receiving increasing attention from software engineering (SE) researchers. Numerous books and papers have been written on the topic (Easterbrook et al. 2008; Glass et al. 2002; Seaman 1999; Singer et al. 2000; Stol et al. 2016b; Wohlin et al. 2012). While these are very

K.-J. Stol (✉)
Lero—the Irish Software Research Centre and School of Computer Science and Information Technology, University College Cork, Ireland
e-mail: klaas-jan.stol@lero.ie

B. Fitzgerald
Lero—the Irish Software Research Centre and Department of Computer Science and Information Systems, University of Limerick, Ireland
e-mail: bf@lero.ie

M. Felderer, G. H. Travassos (eds.), *Contemporary Empirical Methods in Software Engineering*, https://doi.org/10.1007/978-3-030-32489-6_2

useful reference works, there are several issues with the current state of literature on methodology. First, there is a strong emphasis on a limited set of specific methods, in particular experimentation, case studies, and survey studies. Although these are the three most used empirical methods (Stol and Fitzgerald 2018), many other methods exist that have not received the same level of attention. A second issue is that the field has no agreement on an overall taxonomy of methods, which is somewhat problematic as methods vary in terms of granularity and scope. This makes a systematic comparison of methods very challenging. Furthermore, new methods are being adopted from other fields. Grounded Theory, for example, has gained widespread adoption within the SE literature in the last 15 years or so (Stol et al. 2016b). (We note that, like many other methods used in SE, Grounded Theory is not a 'new' method, as it was developed in the 1960s by social scientists Glaser and Strauss—however, its application is relatively new to the SE domain). Other techniques and methods that are relatively new to the software engineering field include the Repertory Grid Technique (Edwards et al. 2009) and ethnography (Sharp et al. 2016). With new methods and techniques being adopted regularly, it becomes challenging to understand how these new methods compare to established approaches. Further, numerous sources present a range of research methods, but these presentations are limited to "shopping lists" of methods: definitions without a systematic comparison. Rather than maintaining a list of definitions of research methods, a more systematic approach is needed that allows us to reason and position existing methods, and new methods as they emerge. Hence, in this chapter we present a taxonomy of research strategies.

There is an additional challenge within the software engineering research community. Different methods have varying strengths and drawbacks, but it is quite common to see unreasonable critiques of studies due to the research methods employed. For example, a common complaint in reviews of case studies is that they do not allow statistical generalizability. Similarly, experiments are often critiqued on the basis that they involved computer science students solving 'toy' problems, thus rendering them unrealistic, and therefore not worthy of publication. Not unreasonably, researchers may wonder which method, then, is the silver bullet that can address all of these limitations?

The answer is none.

Instead of discussing research methods, we raise the level of abstraction and have adopted the term *research strategy*. A research strategy can be considered a category of research methods that have similar trade-offs in terms of generalizability and the level of obtrusiveness or control of the research context—we return to these two dimensions in a later section in this chapter. Previously, we outlined what we have termed the ABC framework of research strategies, and demonstrated how this taxonomy is suitable for software engineering research (Stol and Fitzgerald 2018). In this chapter we draw on this earlier work, elaborate on how the ABC framework is related to Design Science, and provide general guidance for researchers to select appropriate research strategies.

The remainder of this chapter is organized as follows. Section 2 starts with a discussion of research goals, dimensions, and settings. This section presents the

two modes of research, namely knowledge-seeking and solution-seeking research. It outlines the ABC framework, and positions it in relation to Design Science. Section 3 discusses the eight archetypal research strategies that are represented within the ABC framework. For each strategy, we discuss the essence of the strategy, identify a metaphor for the strategy, and provide high-level guidelines. Because research studies are never conducted in isolation, we discuss in Sec. 4 how the ABC framework can be used to design research programmes. Section 5 offers a list of recommended readings. Finally, Section 6 concludes the chapter.

2 Foundations

This section introduces a number of concepts that together form the foundation for the ABC framework. We first introduce the two modes of research in software engineering: knowledge-seeking and solution-seeking research. These are two distinct modes representing different types of activities. This chapter focuses primarily on one mode, namely knowledge-seeking research, but contrasts it with solution-seeking research. In so doing, we draw a link to Design Science. Much has been written on Design Science, which is why we do not discuss it in this chapter. Instead, we refer interested readers to chapter "The Design Science Paradigm as a Frame for Empirical Software Engineering" of this book.

We then return our attention to knowledge-seeking research, and introduce two key dimensions that are present in all knowledge-seeking studies: the level of obtrusiveness and generalizability. Each research strategy represents a unique combination along these two dimensions. This section ends with a discussion of research settings, which refers to the environment in which the research is conducted.

2.1 Knowledge-Seeking vs. Solution-Seeking Research

There are two modes of software engineering research: knowledge-seeking and solution-seeking research. These two modes address different types of questions; Wieringa (2009) has referred to these as knowledge questions and practical problems, respectively. Figure 1 presents how these two modes of research are positioned within the wider context of SE research, the real world, and the SE knowledge base.

Knowledge-seeking studies aim to learn something about the world around us by making observations in some type of environment—this includes the technologies, organizations, and people in natural, contrived, or simulated (virtual) settings. Knowledge-seeking research studies lead to new knowledge, which is typically reported in research papers and books, thereby contributing to the software engineering knowledge base, from which researchers may draw when designing new studies.

In solution-seeking studies, researchers design, create, or develop solutions for a given software engineering challenge. The outcome of these studies include algorithms, models, and tools. Such solution-seeking studies may draw applicable knowledge from the SE knowledge base, which might have originated in either knowledge-seeking or solution-seeking research. Much research within the SE domain is solution-seeking with resulting design artifacts. These artifacts represent "design knowledge," in that they embody knowledge on how a particular engineering problem can be solved—and this knowledge is added to the SE knowledge base as well. Solution-seeking studies fit very well within a Design Science framework (March and Smith 1995; Simon 1996), as discussed in more detail in chapter "The Design Science Paradigm as a Frame for Empirical Software Engineering" of this book. Implemented solutions can be deployed into the real world, and their effectiveness or utility can be studied using knowledge-seeking research. We note that the research process for Design Science as proposed by Hevner et al. (2004) does not align perfectly with solution-seeking research but claims a wider scope that includes evaluation studies—we categorize the latter firmly as knowledge-seeking studies.

As Wieringa (2009) has pointed out, knowledge-seeking and solution-seeking research can be interlinked—nested, even—because knowledge is needed to design solutions, and once designed, a researcher is interested in learning whether the solution works, or how well it compares to other solutions. This linkage is represented by the two white arrows in Fig. 1. In this chapter we are primarily concerned with strategies to conduct knowledge-seeking research, and refer readers interested in

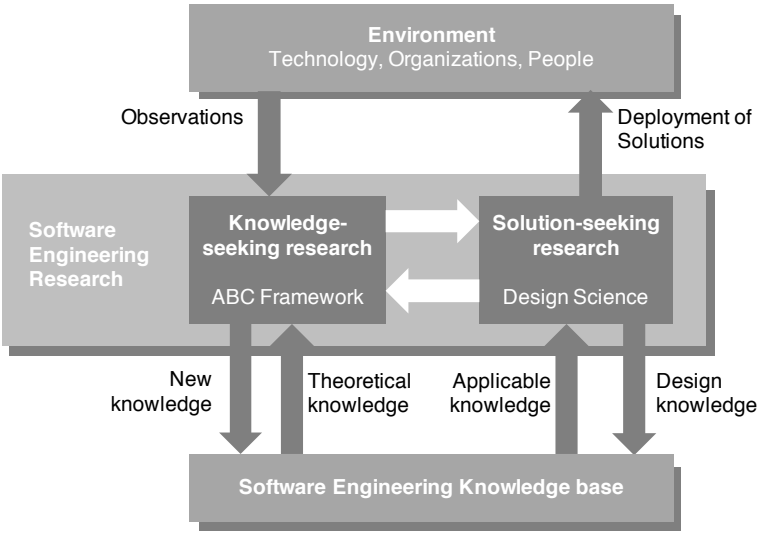


Fig. 1 Knowledge seeking and solution-seeking research: positioning the ABC framework and Design Science

Design Science to chapter "The Design Science Paradigm as a Frame for Empirical Software Engineering."

2.2 Two Dimensions of Research: Obtrusiveness and Generalizability

In the remainder of this chapter we focus primarily on knowledge-seeking research. Numerous methods can be used to "seek knowledge," and as mentioned above, there are numerous sources in the software engineering literature that provide lists of methods. However, a systematic framework to position these methods in relation to one another has been lacking. To address this, we draw on McGrath (1981, 1984, 1995), who organized the most common methods in the social sciences into a methodological "circumplex" that positions eight research strategies. We operationalized the circumplex for a software engineering context, and have labeled the result the "ABC framework" for reasons that will become clear. Below we explain the key concepts of this framework.

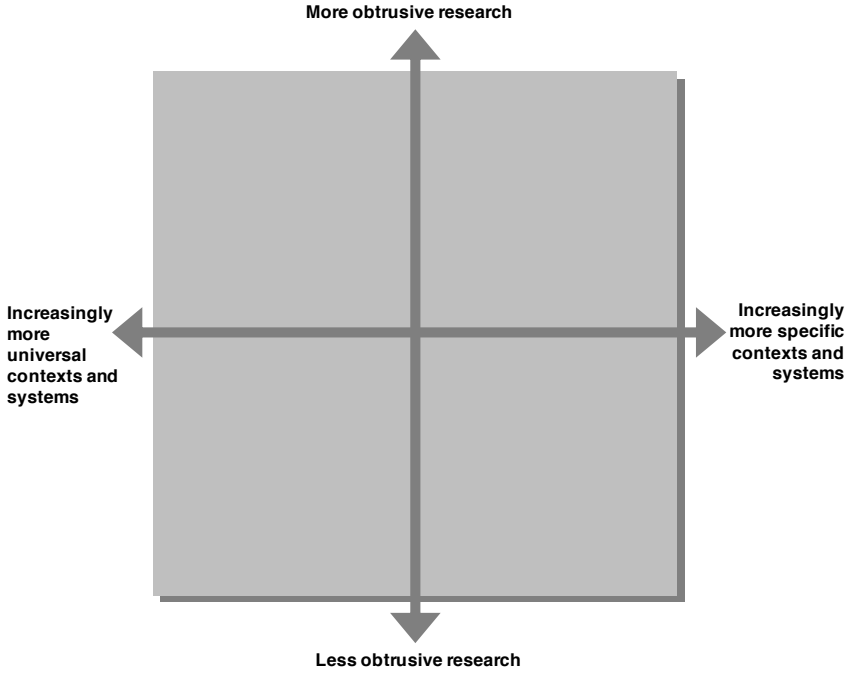


Fig. 2 Two dimensions in knowledge-seeking research

The framework is organized along two dimensions: obtrusiveness and generalizability (see Fig. 2). The first dimension is concerned with how obtrusive the research is: to what extent does a researcher "intrude" on the research setting, or simply make observations in an unobtrusive way. Research methods can vary considerably in the level of intrusion and resulting level of 'control' over the research setting. Clearly, a study that seeks to evaluate the efficiency or performance of a tool requires a careful study set-up, whereas a case study study that seeks to describe how agile methods are tailored at one specific company does not (Fitzgerald et al. 2013).

The second dimension is the level of generalizability of research findings. This is a recurring concern in software engineering research, in particular in the context of case studies. Indeed, exploratory case studies, and other types of field studies, are limited in that the researcher cannot draw any statistically generalizable conclusions from such studies. However, generalization of findings is not the goal of such studies—instead, exploratory case studies and other types of field studies aim to develop an understanding rather than generalization of findings across different settings. Exploratory case studies can be used to theorize and propose hypotheses about other similar contexts.

It is worth noting that a broader view of generalizability beyond that of the statistical sample-based one has also been identified (e.g., Yin (2014); Lee and Baskerville (2003)). Yin identifies Level 1 inference generalizablility which has two forms. The first is the widely known *statistical* generalizability from a representative sample to a population. He also identifies another Level 1 inference, namely from experimental subjects to experimental findings, which is also quite relevant to our research strategies. However, Yin also suggests a further Level 2 inference category of *analytic generalizability* which involves generalizing to theory. This could involve generalizing from a sample to a population, or, indeed, generalizing from field study findings or experimental findings.

2.3 Research Settings

Research takes place in different settings, that is, the environment or context within a researcher conducts research. McGrath (1984) identified four different types of settings to conduct research. Building on the two dimensions described above, these settings are positioned as four quadrants at a 45 degree angle with the main axes that represent the two dimensions described above (see Fig. 3).

The first type of settings is *natural settings*, represented as Quadrant I. Natural settings are those that naturally occur in the 'field' and that exist independently from the researcher conducting research; that is, settings that are host to the phenomenon that a researcher wishes to study. For example, the 'field' for a study on software process improvement is likely to be a software development organization, whereas the 'field' can also be the online communication channels when the topic of study is a particular open source software development project (Mockus et al. 2000)—after all, for open source developers, these online channels are the (virtual) place where

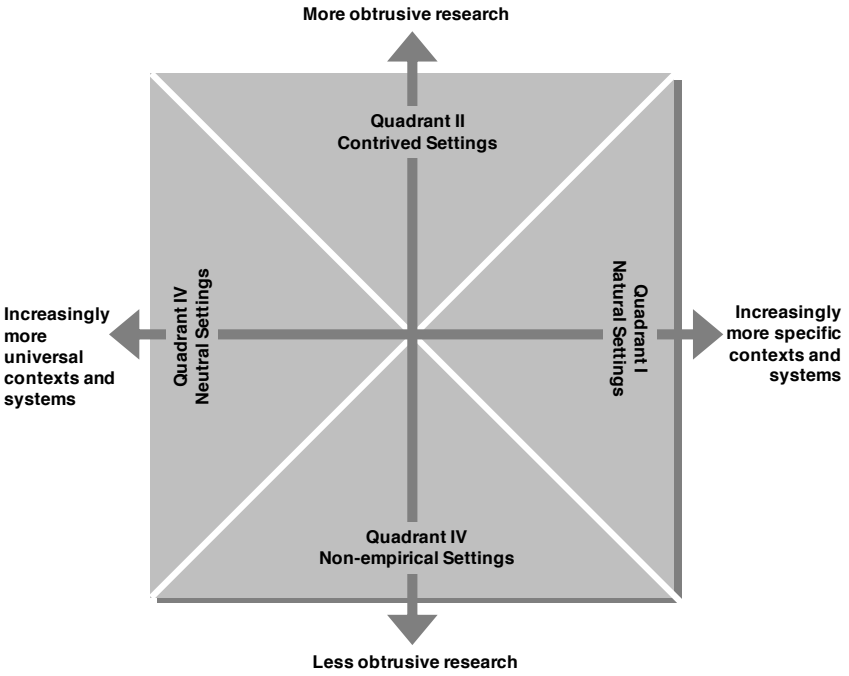


Fig. 3 Research settings in knowledge-seeking research

they communicate and do work. Natural settings are always specific and concrete, rather than abstract and general; hence, the quadrant representing natural settings is positioned on the right-hand side of Fig. 3. Researchers may still exert some level of control over a natural setting (Quadrant I, above the *x* axis), or may simply make empirical observations without manipulating the research setting (below the *x* axis).

In contrast to natural settings, *contrived settings* (represented as Quadrant II) are created by a researcher for the study. In a software engineering research context, contrived settings include laboratories with specific and dedicated equipment to conduct an experiment on some algorithm or software tool. Contrived settings are characterized by a significant degree of control by the researcher. This manifests as the set-up of specialized equipment and measurement instruments that facilitate the execution of a study. Many experimental studies within software engineering are conducted in such contrived settings, whereby algorithms and tools are evaluated for performance and precision. Contrived settings are created by a researcher to either mimic some specific or concrete class of systems (right-hand side of Quadrant II), or a more abstract and generic class of systems (left-hand side of Quadrant II). Either way, a contrived setting is always specifically set up by a researcher, implying the researcher has a high degree of control over the research—hence, Quadrant II is positioned at the upper half of the *x* axis. A contrived setting is essential to conduct

the research—without measurement instruments and other tools such as the design of scenarios or tasks for human participants, the study could not be performed.

There are, however, also studies that do not rely on a specific setting. Some types of studies can take place in *any* setting, and so the setting is *neutral*—this is represented in Fig. 3 as Quadrant III. Researchers may or may not manipulate the research setting; in any case, because the research setting is neutral and not specific to any concrete or specific instance, Quadrant III is positioned at the left-hand side of the *x* axis.

Finally, the fourth type of setting is *non-empirical*, represented by Quadrant IV in Fig. 3). That is, this type of research does not lead to any *empirical* observations. Within software engineering, non-empirical research includes the development of conceptualizations or theoretical frameworks, and computer simulations. While software engineering as a field of study has not traditionally been strongly focused on the development of theory, several initiatives have emerged in recent years to address this (Stol and Fitzgerald 2015; Ralph 2015; Wohlin et al. 2015). Quadrant IV is positioned at the bottom of Fig. 3 because the researcher does not 'intrude' on any empirical setting. Non-empirical research is typically conducted at the researcher's desk or in his or her computer, through the development of symbolic models and computer programs that mimic real settings.

2.4 The ABC of Software Engineering Research

Having laid the foundations for the ABC framework, we now populate the grid in Fig. 3 with eight archetypal research strategies. The result is what we have termed the ABC framework (see Fig. 4). Several of the research strategies will sound familiar, for example Field Study, Laboratory Experiment, and Sample Study, which includes survey studies. Other terms such as Experimental Simulation may be less known within the SE field. Section 3 presents each of these eight strategies in detail. We now turn our attention to the last aspect of the framework, which are the markers A, B, and C.

The term 'ABC' seeks to convey the fact that knowledge-seeking research generally involves actors (A) engaging in behavior (B) in a particular context (C). Within software engineering, actors include software developers, users, managers, and when seeking to generalize over a 'population,' can also include non-human artefacts such as software systems, tools and prototypes. Behavior can relate to that of software engineers, such as coordination, productivity, motivation, and also system behavior (typically involving quality attributes such as reliability and performance). Context can involve industrial settings within organizations, open source communities, or even classroom or laboratory settings.

In the context of our discussion on obtrusiveness and generalizability above, researchers will want to maximize the generalizability of the evidence across actor populations (A), while also exercising precise measurement and control over behavior (B) being studied, in as realistic a context (C) as possible. However,

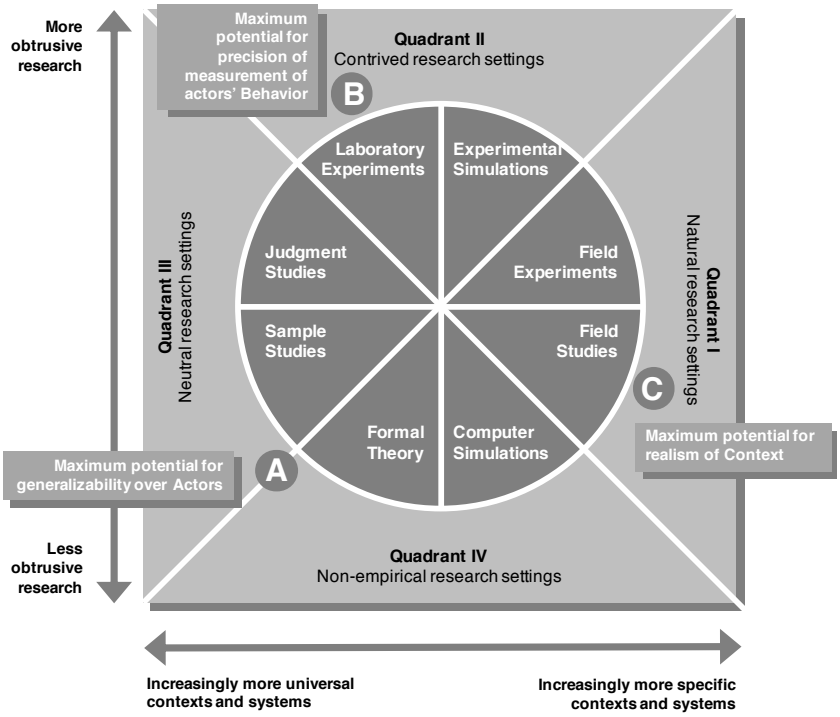


Fig. 4 The ABC framework positions eight archetypal research strategies along two dimensions: generalizability of findings and obtrusiveness of the research (adapted from McGrath (1984)

as McGrath (1981) pointed out, it is impossible to maximize all three goals simultaneously. Increasing precision of measurement and control of behavior (B) for example, inevitably intrudes on and reduces the naturalness and thus the realism of the context (C). Conversely, if one seeks to preserve the realism of context (C), this will reduce both the precision of measurement of behavior (B), and also the degree of generalizability over actors (A). This is reflected in Fig. 4 which identifies the research strategies that are best positioned to deliver for each of the A, B and C. Sample studies can achieve high generalizability (A) but they sacrifice realism of context (C) and precision of behavior (B). Laboratory experiments allow precise measurement and control of behavior (B) but this comes at the expense of the realism of context (C) and generalizability (A). Field studies maximize the realism of context (C) but this is at the expense of control of behavior (B) and generalizability (A). Clearly, the full range of research strategies are required to deliver across all three research goals, and these need to be planned and managed. The above also highlights the fact that certain strategies have inherent and intrinsic weaknesses which cannot be overcome—thus field studies can never provide generalizability, but that is neither their purpose nor strength, and this is not a limitation which can

be overcome when using this research strategy. Therefore, research studies adopting that strategy should not be criticized on that basis.

3 Strategies for Software Engineering Research

In this section we outline the eight archetypal research strategies that are positioned in the ABC framework in Fig. 4. We discuss the research strategies as organized by the quadrants discussed in Sec. 2.3, starting in Quadrant I. Table 1 summarizes the discussion for each strategy, including a metaphor that might help in better understanding the nature and essence of the research strategy, how that setting manifests in software engineering research, and general suggestions as to when to use that strategy.

3.1 Field Studies

Field studies are conducted in natural settings; that is, settings that pre-exist the design of the research study. Field studies are best suited for studying specific instances of phenomena, events, people or teams, and systems. This type of research helps researchers to understand "what's going on," "how things work," and tends to lead to descriptive and exploratory insights. Such descriptions are useful because they provide empirical evidence of phenomena that are relevant to software engineering practitioners, students, and researchers. The findings may provide the basis for hypotheses, which can then be further studied using other strategies. Typical examples in software engineering research are the case studies of the Apache web-server (Mockus et al. 2000) and agile method tailoring (Fitzgerald et al. 2013).

Field studies are relatively unobtrusive with respect to the research setting. The setting for field studies is akin to a jungle, a natural setting that contains unexplained phenomena, unknown tribes, and secrets that the researcher seeks to discover and understand (see Fig. 5). Within a software engineering context, the researcher does not manipulate the research setting, but merely collects data to describe and develop an understanding of a phenomenon, a specific system, or a specific development team. This is why field studies are best suited to offer a high degree of realism of context as the researcher studies a phenomenon within its *natural* setting, and not one that the researcher manipulated.

Typical research methods include the descriptive or exploratory case study, and ethnography (Sharp et al. 2016), but archival studies of legacy systems also fall within the category of field studies, for example, Spinellis and Avgeriou's study of the evolution of Unix's system architecture (Spinellis and Avgeriou 2019). An alternative metaphor for such archival studies is an *archaeological site* rather than a jungle. Data collection methods for field studies include (but are not limited to)

Table 1 Research strategies in software engineering

Strategy	Metaphor	Setting in SE	When to Use
Field Study	Jungle: a natural setting that is ideally left untouched, where creatures and plants can be observed in the wild with a great level of detail.	Software engineering phenomena in a natural context, such as Pair Programming in industry, open source software projects, etc.	To understand phenomena: how does it work? how and why do project teams do what they do? what are characteristics of a phenomenon? Maximum potential to capture a realistic context.
Field Experiment	Nature reserve: a natural setting that has some level of manipulation, e.g. fences, barriers, closed-off sections, sections treated with some intervention.	Industry or open source software projects or teams with some level of researcher intervention; interventions could include different workflows, tools.	To measure 'effects' of some intervention in a natural setting, acknowledging lack of precision due to confounding factors that cannot be controlled for.
Experimental Simulation	Flight simulator: a contrived environment to let pilots train specifically programmed scenarios to evaluate their behavior and decisions. Realism varies depending on resources.	Realism varies from classroom to industry settings designed by researchers with a specific set of tasks or scenarios that recruited participants are asked to process.	To evaluate/measure behavior of participants on a set of tasks in a setting that seeks to resemble a real world setting.
Laboratory Experiment	Cleanroom / test tube: highly controlled setting allowing a researcher to make measurements with high degree of precision.	Classroom or research laboratory settings with a specific set-up and instrumentation to measure e.g. performance of algorithms or tools.	To make high-precision measurements, e.g. for comparing different algorithms and tools. Maximum potential for precision of measurement.
Judgment Study	Courtroom: neutral setting to present evidence / exhibits to a carefully selected panel, asking them for a response (e.g. guilty).	Online or offline setting to solicit input from carefully selected experts after presenting them with a question on a topic or an exhibit (e.g. a new tool).	To get input ('judgment') from experts on a given topic, which requires intense stimulus/response communication.
Sample Study	Referendum: a process to collect a sample of data to seek generalizability over the population. Unusable (invalid) data must be filtered out before analysis.	Online surveys conducted among a population of (typically) developers, or data collected from a software repository. Data must be checked before analysis.	To answer generalizability questions, incl. characterization of a dataset, correlation studies. Maximum potential for generalizability over findings.
Formal Theory	Jigsaw puzzle: attempt to make sense of, integrate, or fit in different pieces into a coherent 'picture.'	Given a set of related observations and evidence regarding a topic of interest, aim to find common patterns and codify these as a theory.	To provide a framework that can describe, explain, or predict phenomena or events of interest, while remaining consistent (generalizable) across different events within some boundary.
Computer Simulation	Weather forecasting system: model of the real world is programmed, capturing as many parameters as possible. Scenarios are run to make informed predictions, but cannot anticipate events not programmed in the simulation.	A computer program that simulates a real world phenomenon, capturing as many important parameters as possible. Program different scenarios to 'run' to explore ranges of, and interactions between, parameters of interest.	To develop an understanding of phenomena and settings that are too complex or expensive to create in the real world.



Fig. 5 Field studies are conducted in pre-existing settings that are not manipulated by a researcher, to study and observe natural phenomena and actors. *Image credits: Public domain*. Source: maxpixel.net (no date)

interviews, document and archival study, or mining repositories—Lethbridge et al. (2005) have discussed a variety of data collection methods and their trade-offs for field studies.

Guidelines for Field Studies

- Use the Field Study strategy to study phenomena in their natural setting, to understand "what's going on," or "how things work." They provide good opportunities to develop substantive theory. Typical methods include the exploratory case study, ethnography, and archival study.
- Field studies require a high level of attention and engagement with the subject and setting. Audio and video recording may help to capture details for later analysis, but these media may affect the behavior of human actors.
- Success of field studies depend on good access to the relevant people and artifacts, which can be particularly challenging within corporate settings. An internal 'champion' or maintaining good relationships is key.
- To generalize findings from field studies, use complementary strategies such as Formal Theory and Sample Studies.

3.2 Field Experiments

Field experiments are also conducted in natural settings, but unlike field studies, this type of study involves some type of manipulation, thus imposing a greater degree of control. That is, the researcher introduces some form of experimental setup by making changes to some variables of interest. After making such changes, the researcher may observe some effect. If field studies are conducted in a 'jungle,' then the setting for a field experiment is more like a *nature reserve*: a dedicated area that may be very similar to a jungle, but the researcher can introduce specific changes to study different aspects within the reserve. Figure 6 shows a jungle with specific patches of trees cut down; the purpose of this study was to study the effects of different types of forest fragmentation on wind dynamics and seed dispersal (Damschen et al. 2014).

It is important to remember that field experiments take place in *natural* settings, which should be clearly distinguished from *contrived* settings that provide the setting for experimental simulations and laboratory experiments, discussed later.

A range of methods are available to conduct field experiments. These are not limited to the traditional controlled experiment that separates a population of actors into two or more different groups so as to make comparisons. Experimentation also occurs when a researcher adopts the Action Research method; with Action Research, a researcher follows a recurring cycle of making changes and evaluating the results of those changes. While not a randomized controlled experiment, Action Research can still be considered a form of experimentation.

Ebert et al. (2001) conducted a field experiment to investigate three factors that might impact the cost of rework in distributed software development: (1) the effect

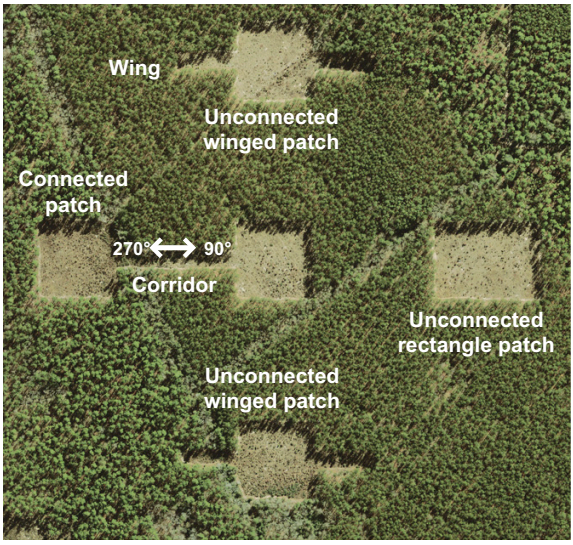


Fig. 6 Field experiments involve the manipulation of an otherwise pre-existing natural setting to facilitate observation and measurement in order to collect data. *Image source:* Damschen et al. (2014), used with permission.

of co-location on the efficiency and effectiveness of defect detection; (2) the effect of coaching on software quality, and (3) the effect of changes to the development process on teamwork, and continuous build on management of distributed project. To evaluate these effects, Ebert et al. used project data that the company had gathered for several years.

Despite careful measurement of a range of parameters, certain factors are hard to measure, such as 'culture.' Ebert et al. divided the projects into different sets, e.g., "within one culture (i.e. Europe)." While we can certainly generalize that there are common attributes across different European cultures, there is no single Europe culture—each European culture is quite distinct, with significant changes even between neighboring countries such as Ireland and the UK, or the Netherlands and Germany. Furthermore, each of the projects in the data set used by Ebert et al. will undoubtedly have had specific obstacles, such as particularly challenging technology or changing requirements, and strengths such as particularly talented staff. These factors are very hard, if not impossible to capture, reducing the precision of measurement.

Guidelines for Field Experiments

- Use the field experiment strategy to evaluate the effects of manipulations within realistic settings.
- Field experiments require a high degree of prior investment in design and execution of data collection, typically in corporate settings.
- Contextual factors must be recorded carefully, so that they can be considered in the analysis.
- Use Formal Theory or Sample Studies to seek a higher degree of generalizability. Computer simulations can be used to model a complex real-world system to explore key parameters and their interactions, when a field experiment would be too costly.

3.3 Experimental Simulations

Experimental simulations combine some elements of the field experiment strategy and laboratory experiments (discussed below). A key difference with field experiments is that experimental simulations take place in contrived settings. That is, the research environment is artificial and is purposely designed to conduct the study—before and beyond the study, it does not exist. While this makes the experimental simulation less realistic, it also gives the researcher opportunities to make more precise measurements and observations, because participants can be asked to do specifically designed tasks that may not be part of their daily routine. This increases the level of control even further with respect to field experiments.



Fig. 7 Flight simulators are experimental simulations that facilitate training and study of the behavior of pilots in pre-programmed scenarios. *Image credits: SuperJet International, distributed under CC BY-SA 2.0* Source: SuperJet International (2011)

While we compared the field experiment to a nature reserve, the experimental simulation is more akin to a greenhouse, which mimics a warmer climate. The researcher is still interested in natural processes (e.g. how do flora flourish), but the setting in which that process is observed is artificially created for that purpose.

The greenhouse metaphor links well to the jungle and nature reserve metaphors, but another useful metaphor for the experimental simulation is a flight simulator (see Fig. 7). Within the flight simulator, specific events can be introduced, such as heavy storms and rainfall. Pilots in training would be asked to perform as they would in a real aircraft, but the research setting is considerably easier and cheaper to plan.

The level of realism that is achieved in experimental simulations can vary considerably, just like flight simulators. The latter can vary from low-cost set-ups consisting of a standard PC, a budget flight yoke and rudder pedals, to high-end, full motion flight simulators used by professional pilots that might cost millions of dollars. The tasks that participants are asked to perform in experimental simulation may also vary in realism. While such tasks are part of normal daily life of the participants in field experiments, in experimental simulations participants are recruited and invited to perform certain tasks designed by the researcher. The process that the researcher wishes to study is simulated, facilitating systematic measurement and comparison. The level of realism of the task can be very high, such as debugging a program within a professional setting (Jiang et al. 2017), or it

can be as contrived as producing and trading colored shapes, such as blue squares and red circles (Bos et al. 2004).

Jiang et al. (2017) conducted an experimental simulation to investigate whether developers conduct impact analysis during debugging. Their contrived setting consisted of a specifically set up work station, equipped with the SimpleScreenRecorder recording software. Videos were captured of nine professional developers who had been given two bug reports. The bug reports had been identified by the researchers in two specific applications (PdfSam and Raptor), and were selected because these bugs had already been fixed at the time of the study. While the task at hand and the study setting were contrived (designed by the researchers), both were quite realistic. However, the realism of the study was reduced as the researchers set a time limit for some of the participants, and offered a suggested fix to the defects.

In contrast, Bos et al. (2004) focused on collaboration between co-located and distributed individuals, and the task at hand was simply a mechanism to engage participants. The focus of the study was therefore on the emergent behavior of the participants, rather than how a specific task was performed as was the case for the Jiang et al. study described above. Therefore, to ensure that the participants understood the task at hand, and to minimize any distraction that might ensue from introducing complex tasks, the researchers chose to design trivial tasks.

Guidelines for Experimental Simulations

- Use experimental simulations to find an appropriate balance between measuring emergent behavior and achieving realism.
- Choose an appropriate level of realism of the research setting, balancing between cost and effort of setting up the research setting and the need for realism.
- If the research focus is on participant behavior that is not dependent on a specific type of task, then minimize the complexity of the task.
- If the research focus is on the behavior in relation to a specific type of task, then the task should exhibit a high level of realism.

3.4 Laboratory Experiments

The laboratory experiment is a strategy that offers maximum opportunity to control the research setting. Like experimental simulations, laboratory experiments take place in contrived settings that are created by the researcher. Laboratory experiments involve a careful manipulation or initialization of variables and settings that give the researcher the maximum opportunity to take precise measurements of actors' behavior, whether these are human participants or software systems.

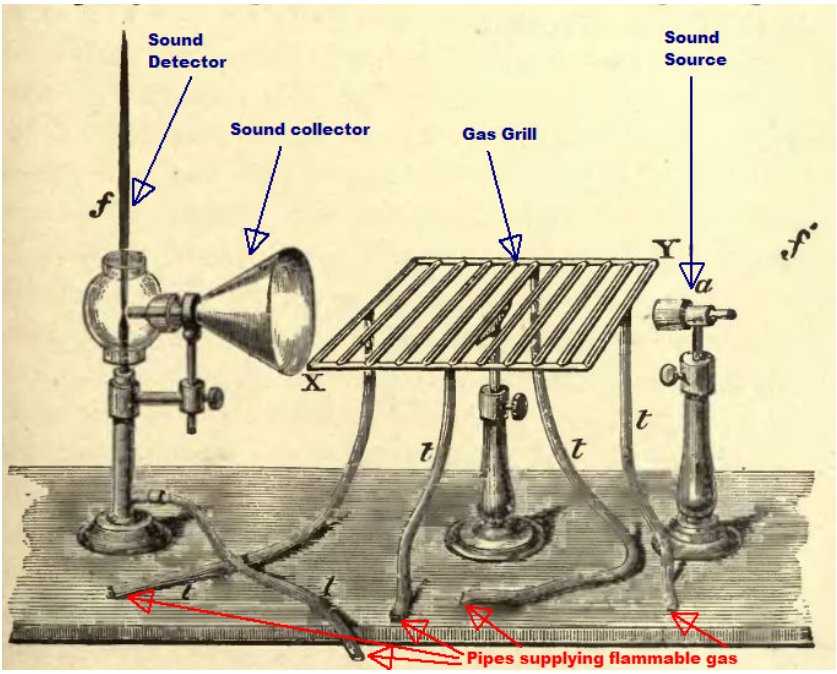


Fig. 8 Laboratory experiments are conducted in settings contrived by the researcher to allow for a maximum level of precision of measurement. *Image credits: Public domain*. Source: Tyndall (1896)

If the experimental simulation is akin to a greenhouse, then we can compare the laboratory experiment to a test tube or cleanroom (see Fig. 8). Both a greenhouse and a test tube provide a contrived setting, but the difference lies in the compromise that the researcher makes between the level of control over the setting on the one hand, and the level of realism on the other hand.

Laboratory experiments can be conducted with human participants or with programmed 'actors': algorithms, prototype tools, etc. A laboratory experiment with human participants usually involves a treatment and a control group, and the researcher aims to measure with a high degree of precision to observe any differences. A good example of this is a study by Niknafs and Berry (2017), who investigated the impact of domain knowledge on the effectiveness of requirements engineering activities. Niknafs and Berry conducted two controlled experiments involving a total of 40 teams of three members each. The participants were all technological students (computer science, software engineering, and other technical areas). Their article offers a detailed presentation of the experimental setup, procedures, and variables that they measured.

Programmed actors are systems running specific algorithms or other software applications that are experimentally evaluated. This approach is extremely common in software engineering research studies, because a significant portion of SE studies

offer new solutions in the form of new algorithms and tools which are subsequently evaluated (Stol and Fitzgerald 2018).

It is worth noting that many software engineering papers characterize the laboratory experiments they report as "real-world case studies," but this is a misrepresentation. The mere fact that a realistic system or operational data is used does not mean that such studies exhibit a higher degree of realism—such studies are still laboratory experiments, not field experiments or field studies. While there is clearly value in engaging with industry partners to access their data or source code for an experimental study, we observe that such studies are often misrepresented, presumably to convince the reader of the relevance and rigor of the work.

An example of a laboratory experiment using programmed actors is Li et al. (2017)'s evaluation of search algorithms. Li et al. proposed a new fitness function to address an optimization problem, and evaluated the newly proposed algorithm with commonly used search algorithms. Li et al. offer a detailed description of the experimental setup and the statistical analyses that were conducted on the collected data. This study clearly took place in a setting specifically contrived, aiming to maximize the precision of measurement so as to be able to conclude which algorithm performed best. Such comparative studies are very common in software engineering, and are sometimes referred as benchmarking studies (Sim et al. 2003).

Guidelines for Laboratory Experiments

- Use laboratory experiments to achieve a high level of precision in measuring variables.
- Laboratory experiments may involve either students or professionals; the use of professionals does not make a study a field experiment *per se*.
- Laboratory experiments may involve the use of data or systems that come from natural settings, but this does not make such studies field experiments.
- To address the limitation of a lack of realism, laboratory experiments can be complemented with experimental simulations and field experiments.

3.5 Judgment Studies

Judgment studies are positioned next to the laboratory experiment. The latter allow maximum precision of measurement in a research setting that is fully under the control of the researcher, who plans and introduces stimuli into the research setting. Judgment studies also involve stimuli introduced by the researcher, but rather than observing or measuring behavior, the researcher is interested in the *responses* of the participants. Judgment studies take place in neutralized settings—the researcher does not need a contrived setting, but can conduct the study in any setting that



Fig. 9 Judgment studies are usually conducted in neutral settings so that participants can focus on the study—much like how a case is presented in a courtroom. *Image credits: Fayerollinson CC BY-SA 3.0*, Source: Fayerollinson (2010)

is available. The primary consideration is that participants are not distracted. For example, if a judgment study takes place in settings that make the participants uncomfortable due to factors such as temperature or noise, then this may inhibit completion of the study. Hence, the researcher may actively seek to neutralize the setting—much like a *courtroom* (see Fig. 9). Rather than manipulating the study setting, the researcher conducts the study by carefully and systematically selecting appropriate participants (or 'judges').

Judgment studies commonly include Delphi studies and focus group studies. These methods involve systematic and purposively selected experts whom the researcher deems to be suitable for the study. These studies usually do not involve a large number of participants—that is, judgment studies do not seek statistical generalizability over the population of which the experts are representatives, but rather generalizability over the experts' *responses* (McGrath 1984).

One example of a judgment study is Krafft's (2016) Delphi study that investigated how open source developers pick their tools, when they contribute to a large project and thus where tools must be compatible. Krafft's study involved a systematic selection of 24 experts within the Debian open source community, which were asked for their input in three different rounds. Each round, the data were analyzed, aggregated, and sent back to the panel to solicit further input. The setting of this study was not important; in fact, the study was conducted mostly via email interaction, and so the experts were simply based in the comfort of their own home or office space. Rather than generalizing over the population of OSS developers, Krafft sought to develop a generalizable answer to his question, namely, how do Debian developers select their tools.

There are of course studies that seek generalizability over a population. These are sample studies, and we discuss them next.

Guidelines for Judgment Studies

- Judgment studies represent a compromise between a sample study (generalizability) and a laboratory experiment (precision of measurement): use judgment studies to seek generalizability over participants' *responses*, not the participants themselves.
- Judgment studies rely on a systematic sampling rather than representative sampling; careful selection of experts is essential.
- Findings from a judgment study can be evaluated through large-scale sample studies, or categories of observed behavior or responses can be further studied in realistic settings through field studies.

3.6 Sample Studies

Sample studies involve the collection of data from a population, whether that population consists of human actors or of system artifacts. Surveys are useful to seek generalizability over a population by studying a sample of that population. For example, researchers who wish to learn more about open source software developers could conduct a sample study among developers active on public GitHub projects. Sample studies are also discussed in chapter "Challenges in Survey Research" of this book.

Like judgment studies, sample studies do not depend on a specific research setting that is set up by a researcher. The setting plays no role in the research; no specific setting is required to conduct sample studies.

We compare a sample study to a referendum or election (see Fig. 10), with the goal of collecting a relatively small number of data points from a large number of actors. We use the abstract term 'actor' here, because actors may be human participants (as in a referendum), but sample studies are also widely used in SE research by collecting data from software repositories. In sample studies that collect data from human respondents, a critical issue is whether or not a sufficiently large sample can be collected, because response rates tend to be limited. Rates of less than 30% are not unusual, and this can make survey research quite challenging. (We note that it is not possible to calculate a response rate if the size of the target group is unknown).

Sample studies have high potential to achieve generalizability of findings to a larger population of actors, whether they be software developers or software system artifacts such as bug reports. However, sample studies are inherently limited in that they offer limited precision in measuring behavior, either human or system



Fig. 10 Sample studies with human participants are like referendums or elections; the amount of information gathered per participant is limited, and achieving a good response rate can be challenging if the population has low interest. *Image credits: Australian Electoral Commission, distributed under CC BY 3.0*. Source: Australian Electoral Commission (2016)

behavior. The reason for this limitation is that the researcher collects whatever data he or she can get. For example, even a carefully designed survey instrument aimed at collecting data from software professionals may still be misinterpreted by respondents, or respondents may accidentally or deliberately skip questions. The researcher's control over this is limited. When collecting data from a software repository, the researcher can only gather the data that is stored, which is not always what the researcher would like to have. Furthermore, data in software repositories may not be consistent or correct. Indeed, many studies have investigated events on the popular GitHub.com development platform, but mining that repository is not free from perils (Kalliamvakou et al. 2016). In some cases, database tables and fields may be ambiguously named and labeled, leading to misinterpretations by the researcher relying on the data for analysis.

Sample studies are among the most common studies in software engineering (Stol and Fitzgerald 2018), though the type of data analysis can vary widely. Quantitative data analysis types vary from descriptive to inferential (Russo and Stol 2019). Storey et al. (2017)'s sample study of GitHub developers sought to understand developers' use of communication channels, and used a descriptive analysis. An example of a sample study using inferential analysis is Sharma and Stol (2020)'s study of onboarding of software professionals that sought to understand what makes for a successful onboarding experience of new hires.

An example of a sample study of software development artifacts is Stol et al. (2017)'s analysis of crowdsourcing contests. This study investigated the potential

influence of a number of factors on the interest and participation of members of the crowd in contests, based on a sample of over 13,000 crowdsourcing contests on the Topcoder platform.

Guidelines for Sample Studies

- Use sample studies to achieve maximum generalizability over a population, whether that is a population of human actors (e.g. software developers, managers), or software actors or artifacts (e.g. bugs and bug fixes, apps, projects).
- The number of data points per subject is usually limited, and so the questions must be carefully selected.
- As there is often no direct contact between the researcher and human respondents, questions must be unambiguous.
- When relying on archival data from software repositories, the data comes 'as-is'; sanity checks must be conducted to ensure consistency.
- Generalizability is limited by the sample that is studied.

3.7 Formal Theory

Formal theory is a strategy that aims to seek generalizability, not through empirical methods, but rather through the specification of symbolic representations of variables and constructs (Runkel and McGrath 1972). Therefore, formal theory takes place in a non-empirical setting. Formal theory development typically involves extensive reviews of prior research in order to identify, distill and codify recurring patterns. Thinking in terms of abstractions and aiming to identify theoretical relationships is one of the most important activities in research. However, the amount of prior research may be quite scarce, and indeed, theories can be proposed on little more than a rich imagination and mental models that develop over time. The importance of theory development is illustrated by Nisbett (2005, p. 4), who described how the early Mesopotamian and Egyptian civilizations made systematic (empirical) observations, but only the Greeks made significant progress by explaining their observations in terms of the principles underpinning them—that is, by reasoning what might link or cause those observations.

A major role of theory is to inform future research, as it motivates further studies to evaluate hypothesized relationships—this is true both for the Periodic Table of Elements and for Einstein's Theory of Relativity. Both theories allowed predictions to be made which could only be empirically verified many decades later.

The process of formal theory is like making a jigsaw puzzle (see Fig. 11). When you first open up the box, there may be many pieces, and it may not be readily



Fig. 11 Formal theory can be similar to making a jigsaw puzzle: lots of pieces, and a challenge to put it all together in a coherent way. *Image credits: public domain.*

clear as to how they all fit together.[1] This is the case in many areas of software engineering: much empirical research exists, but the field lacks theories that can explain and integrate these individual studies. Of course, theories may also be developed without much initial empirical evidence. Einstein's proposed theory of relativity was not grounded in any empirical observations, but rather through *"sheer genius, fully formed from the mind of the theorist"* (Hassan 2015). Empirical evidence for Einstein's theory has been gathered since.

Both the Theory of Relativity and the Periodic Table of Elements are "general" or "grand" theories, as they are *"all embracing, unified theories"* that are relatively unbounded (Hassan 2015). Formal theory may refer to grand theory, but may also refer to middle range theories—those that are more limited in scope and context (Bourgeois 1979).

It is worth clarifying how Grounded Theory (GT) relates to formal theory as a research strategy. GT is an approach by which the researcher makes empirical observations, typically through field studies, and in parallel generates theory that is grounded in those observations. Barney Glaser, one of the two creators of GT together with Anselm Strauss (Glaser and Strauss 1967), explicitly refers to such theory as "substantive theory" distinguishing it from "formal grounded theory" (Glaser 1978). Glaser (1978, p. 52) describes the difference between substantive and formal grounded theory as:

> The former is about a specific area e.g., route milkmen, the latter about a concept in its full generality: e.g. cultivating.

[1] This is also the point where the jigsaw puzzle metaphor breaks down, as jigsaw puzzles tend to come in a box with the solution printed on the cover—researchers do not have such luxury.

Thus, the distinction that Glaser draws here is that substantive theory explains some specific phenomenon (e.g. a milkman cultivating relationships with his customers), while formal theory is at a higher level of abstraction (e.g. the act of cultivating relationships in *any* type of setting).

Formal theory is positioned on the left-hand side of the ABC framework, because it refers to those theories that have a sufficient degree of generalizability and are not intrinsically linked to any substantive domain. Using the example above, a substantive theory of a milkman cultivating relationships with his customers would not exhibit sufficient generalizability, while a formal grounded theory explaining the general act of cultivating relationships would do so.

In software engineering, much "theory" is what we have termed "micro-theories" (Stol et al. 2016a)—and what Merton would call *"the minor but necessary working hypotheses that evolve in abundance during day-to-day research* (Merton 1968, p. 39). Other forms of theory can also be observed, such as theoretical frameworks, models, or other types of conceptualizations (Stol and Fitzgerald 2015). The traditional forms of theories found in the social sciences (variance, process theory) are less common in software engineering research, though this has started to change in recent years. While 'formal theory' (i.e. general theory) by Glaser's description is not common in software engineering, we believe generalizable conceptualizations should still be an ambition of the community.

One example of what we would classify as formal theory is Ralph's theory of Sensemaking, Coevolution, and Implementation (SCI) (Ralph 2015). SCI seeks to *"replace lifecycle depictions of the development process"* because the latter suggest software design as a linear sequence of phases and label these phases as mutually exclusive activities. SCI offers a new perspective that researchers and educators may adopt to study and teach software development processes.

Arguably another form of substantive theory is design artifacts that are the result of the Design Science paradigm. Such products are also "vehicles" of knowledge. Runeson et al. discuss the Design Science paradigm in software engineering research in chapter "The Design Science Paradigm as a Frame for Empirical Software Engineering" of this book.

Guidelines for Formal Theory

- Develop formal theory when an area of interest attracts a high number of empirical studies without an overall framework to integrate the findings.
- Formal theory is also a useful starting point *before* conducting any empirical studies as it helps to focus and identify important research questions or hypotheses, or to predict observations (e.g. the Periodic Table of Elements).
- Formal theory can be developed based on previous empirical observations (for example through field studies), substantive theory (identified through for example literature reviews) and computer simulations.

3.8 Computer Simulations

A computer simulation is a fully closed system that implements a concrete theoretical model. One type of computer simulation that people living in rainy climates can appreciate is weather forecasting systems (see Fig. 12). Predicting the weather is done by means of highly complex computer models that are carefully configured and calibrated using a wide range of parameters. By considering a series of scenarios, the most likely scenario will then be the basis for any forecasts. These weather forecasting systems are completely closed, in that *all* parameters and equations are fully programmed. The values of these parameters may be based on values empirically observed through a range of sensors throughout the country, but once these are entered into the simulation, there is no further interaction with the outside world—computer simulations do not make any new *empirical* observations. Or, as McGrath stated succinctly: *"no new behavior transpires during the run of the simulation"* (McGrath 1995, p. 159). This is one key characteristic that sets computer simulations apart from experimental simulations.

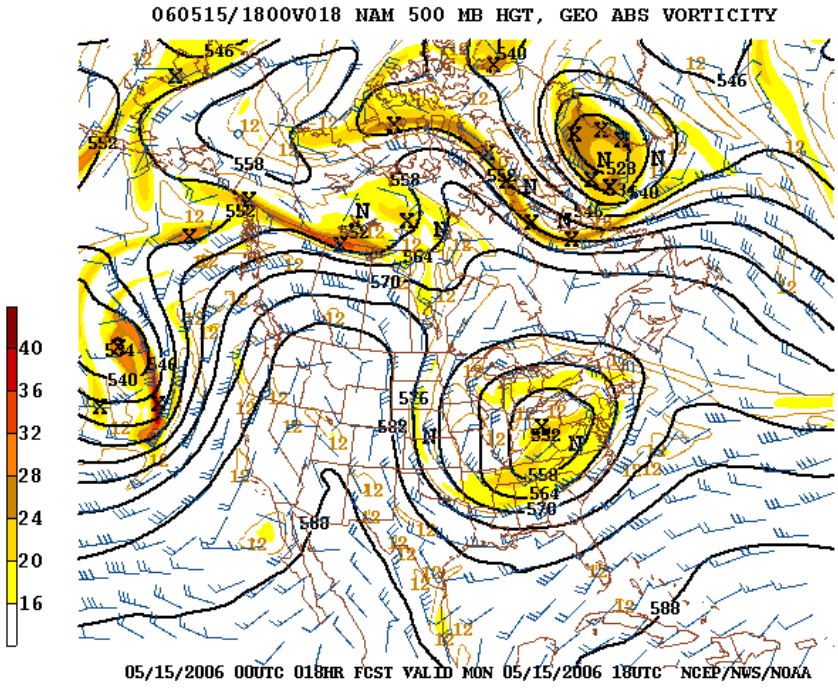


Fig. 12 Computer simulations are like weather forecasting systems. Computer models take empirical observations and a set of complex mathematical models to run scenarios. The result is a forecast: information, but not empirically gained, and may be imprecise. *Image credits: public domain*

Because computer simulations do not make any empirical observations, the results from a simulation should be treated with care. Any predictions or results coming from a computer simulation might be wrong—while weather forecasting computers can make impressively precise predictions, occasionally they are still wrong. This is because computer simulations are based on models of reality, not reality itself.

A good example of the use of computer simulations is a study to evaluate task allocation strategies in distributed software development (Setamanit 2007; Setamanit et al. 2007). Software development tasks in distributed settings can be allocated following three strategies: Follow-the-Sun (FTS), phase-based, and module-based. Using FTS, one team may finish the workday, as another team located elsewhere may start the workday. The work continues potentially 24/7, depending on the number of teams and the time differences between them. Phase-based development means that each team takes responsibility for a particular phase of the development lifecycle. Module-based development implies that each development team has end-to-end responsibility for a given software module. By running a number of scenarios with computer simulations, Setamanit et al. found that by neutralizing any communication and cultural barriers, the FTS strategy led to a development cycle that was 70% shorter than a single-site development scenario. However, when these communication and cultural factors were introduced into the model, the FTS strategy performed considerably worse than single-site development.

It is important to realize that the scenarios are modeled based on formulas and assumptions, and that the outcome of these computer simulations may not correspond to reality. Chapter "The Role of Simulation-Based Studies in Software Engineering Research" of this book discusses computer simulations in more detail.

Guidelines for Computer Simulations

- Use computer simulations to create a model of real-world systems or phenomena that cannot be easily or affordably set-up in real life.
- Identify and model all parameters of interest, and which have relevance as suggested by prior literature.
- Be cautious in interpreting the results of computer simulations as they are necessarily simplified models of reality. It is important to remember that computer simulations do not generate *empirical* observations.
- Conduct empirical studies to confirm or disconfirm the results of computer simulations.

4 Applying the ABC Framework

Research studies are rarely conducted in isolation, but usually as part of a research program that seeks to investigate a phenomenon. This is true for PhD dissertations, but also for funded research programmes, such as those funded by the US National Science Foundation (NSF), the European Committee's funding programmes such as Horizon 2020 and its follow-up Horizon Europe, or other funding programmes. In order to study a phenomenon, it is useful to employ different research strategies—each strategy has potential strengths and inherent limitations, and by using different strategies to study the same topic, researchers can address such inherent limitations, and ultimately learn more about the topic of study. In our previous work, we discussed two scenarios (Stol and Fitzgerald 2018). To complement those, we discuss a recent research programme that we were both involved in.

In Fig. 13 we present the positioning of strategies used by Dr. Ann Barcomb's PhD dissertation, which we co-supervised (together with Prof. Dirk Riehle at the Friedrich-Alexander University Erlangen-Nürnberg) (Barcomb 2019). Ann's dissertation, entitled *"Retaining and Managing Episodic Contributors in Free/Libre/Open Source Software Communities"* (Barcomb et al. 2019a), focused on episodic volunteers: those volunteers that may contribute intermittently.

The dissertation was designed as a set of three empirical studies. The first study was a qualitative survey that sought to document what episodic volunteering looks like in an open source software setting (Barcomb et al. 2019a). The study involved interviews with members of 13 different open source communities. The choice of an exploratory survey is interesting. While the field study strategy seems a straightforward choice for topics that have not been studied in great detail, this

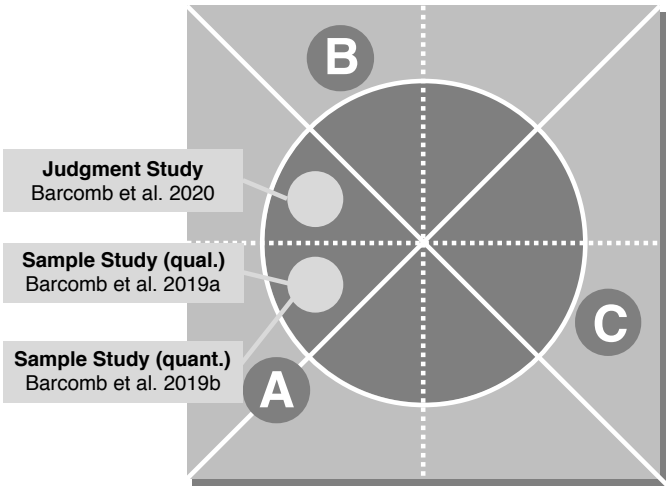


Fig. 13 Three studies of Ann Barcomb's PhD dissertation: two sample studies and one judgment study.

study sought a higher degree of generalizability to achieve the research goal. If only a single OSS project had been studied, for example by means of a case study or ethnographic study, the findings would have been rather limited: we would have learned in great detail, and enriched with great contextual detail how episodic volunteers behave or operate in one project, but this would not have answered the question *"what does episodic volunteering look like in OSS projects?"* Thus, a qualitative survey was deemed a more useful approach.

The second study drew on the general literature on volunteering and episodic volunteers (Barcomb et al. 2019b). Specifically, other researchers had developed a theoretical model that sought to explain reasons why episodic volunteers might return. Ann's second study therefore sought to test this theoretical model in the context of OSS episodic volunteers. This study consisted of a survey instrument implemented with SurveyMonkey; 101 usable responses analyzed using PLS structural equation modeling.

Having established (1) what episodic volunteering looks like in an OSS setting, (2) what reasons might help to retain episodic volunteers in OSS communities, Ann's next focus was on managing those episodic volunteers. Hence, the third study sought to identify practices that OSS community managers believe are useful to do so (Barcomb et al. 2020). Through a Delphi study, Ann carefully selected 24 community managers from a variety of communities, and engaged them in three rounds of interaction (through email). The Delphi study is categorized as a Judgment Study: each of the panel members was asked to provide detailed responses to a set of carefully crafted questions. In each of the three rounds, responses were analyzed, anonymized, and collated. At the end of the three rounds, the analysis resulted in a set of practices, based on community experts' input, for managing episodic volunteers.

Together, these three studies add considerable insight to this nascent area within the larger OSS literature. The selected research strategies are all positioned on the left-hand side of the ABC framework (see Fig. 13), but still vary in certain aspects. While Ann conducted two sample studies, one was of qualitative nature, whereas the other was quantitative. The third study is a judgment study, which provides a higher degree of control of the conversation—indeed, the Delphi method facilitates interaction in several rounds, providing researchers an opportunity to clarify answers when needed; such flexibility was not available in the quantitative sample study.

It is interesting to reflect on the design of this research programme (Table 2). The programme did not involve any experimentation, nor did it involve any field studies. While one of the studies involved face-to-face interviews, the setting in which these took place was of no importance—the goal was not to focus on the context of a specific volunteer in a specific project. Arguably this is one limitation of the research programme, which could be addressed in the future by conducting a case study or ethnographic study of episodic volunteers in a specific project. This could potentially lead to rich insights as to the reasoning process on a day-to-day basis of episodic volunteers whether or not to contribute or return to the project they have been involved in before.

Table 2 Summary of the three studies in Barcomb's dissertation

Study	Strategy	Description	Limitations
1	Sample study	Qualitative survey involving interviews with 20 informants from 13 OSS communities selected based on variety across two dimensions.	Captures a range of perspectives on episodic volunteering in OSS communities, but does not establish any causal relationships or any specific context.
2	Sample study	Cross-sectional, quantitative survey with 101 responses to evaluate a theoretical model.	Seeks generalizability, but does not capture any causal relationships or any specific context.
3	Judgment study	Delphi study involving a panel of 24 experts; documents a set of practices to manage episodic volunteers.	Offers a trade-off between generalizability and precision of measurement; does not capture any specific context.

5 Recommended Further Reading

This chapter provides a high level framework to help researchers in their selection of an appropriate knowledge-seeking research strategy. What this chapter does not offer is detailed guidance for each and every specific method and technique that can be positioned within the framework. References to a wide range of excellent resources are provided in a previous article (Stol and Fitzgerald 2018). In this section we list a number of recommended sources organized by theme or research strategy.

5.1 Empirical Studies in Software Engineering

Numerous sources discuss general matters regarding empirical studies which we cannot list all. We suggest the following as a starting point.

Glass et al. (2002) were among the first to reflect on the research methods used within the software engineering research community through an extensive literature review, and observed little variation in research approach and methods methodological. Much has changed in the two decades that have passed. Kitchenham has written extensively on empirical software engineering since the late nineties. She and her co-authors have primarily focused on quantitative methods, including survey research (Kitchenham and Pfleeger 2002) and experimentation (Kitchenham et al. 2002). In 2004, Kitchenham et al. (2004) published a seminal paper arguing for evidence-based software engineering, which borrows from the concept of evidence-based medicine, arguing that software engineering practice should be informed by evidence. A key source for many has been Easterbrook et al. (2008)'s guidelines for selecting empirical methods, which provides an overview of several

widely used methods as well as a brief discussion of epistemology within software engineering. In the decade since, several other works have discussed the maturity of empirical software engineering—a recent article by Méndez Fernández and Passoth (2019) discusses the increasing focus on human-centric challenges and the need to establish interdisciplinary collaborations.

5.2 Field Studies

Numerous research methods fall within the scope of the field study strategy. Most common among those is the descriptive or exploratory case study. A widely cited resource is Yin (2014)'s book. Runeson et al. (2012) have tailored guidance for case studies to the software engineering domain. Lethbridge et al. (2005) have presented a taxonomy of data collection techniques for field studies, whereas Seaman (1999) presented a seminal paper on the use of qualitative methods, which are typically used in field studies. Besides the case study method, two other methods warrant brief discussion. The first is Grounded Theory, a method originally proposed by Glaser and Strauss (1967), and which has since seen more specific interpretations (Stol et al. 2016b). Grounded Theory studies have become common in software engineering, though in many cases the term has been misused (Stol et al. 2016b). We note that Grounded Theory studies do not always focus on one specific research setting, but could also be used, for example, to investigate a range of different companies (cf. Hoda et al. (2013)). Another method within the strategy of field studies is the ethnography; Sharp et al. (2016) have discussed its role within the software engineering domain.

5.3 Experimental Studies

There are numerous sources that provide advice for experimental studies. Besides Kitchenham et al. (2002)'s preliminary guidelines, we suggest interested readers consult Wohlin et al. (2012)'s discussion of experimentation in software engineering as well as Juristo and Moreno (2001)'s book on the same topic.

5.4 Judgment Studies and Sample Studies

Judgment studies, sometimes referred to as *user studies*, can be a useful way to evaluate a tool or get insights from a carefully selected set of experts. Focus group studies and Delphi studies are also methods that fit clearly within this strategy. The Delphi method has been well documented by Dalkey and Helmer (1963) and Linstone and Turoff (2002). Kontio et al. (2008) offer guidance for focus group

studies. Another method that fits well within this strategy is the repertory grid technique; Edwards et al. (2009) discuss its role within software engineering.

Sample studies are among the most common in software engineering (Stol and Fitzgerald 2018). Kitchenham and Pfleeger (2002) have published a six-part series of guidelines in ACM Software Engineering Notes. Chapter "Challenges in Survey Research" in this book also discuss sample studies. Studies that use samples of development artifacts from software repositories such as GitHub are extremely common as well. Kalliamvakou et al. (2016) offer useful guidance to address the many pitfalls in such studies.

5.5 Formal Theory and Computer Simulations

An important category of research that is often overlooked due to the strong focus on empirical methods is non-empirical research. The two strategies defined in the ABC framework, Formal Theory and Computer Simulations, are both useful approaches that can complement empirical methods in a variety of ways. In earlier work, we coined the concept of theory-oriented software engineering, emphasizing that research studies consist of elements from three different 'domains': the substantive domain, representing some phenomenon of interest, the methodological domain, representing the variety of methods to study that phenomenon, and the conceptual domain, which represents any theoretical construct or framework to design a study or make sense of its findings. While most researchers are familiar with so-called variance theories, which seek to link different measurable constructs, Ralph (2018) provides methodological guidelines for so-called process theories. It is worth noting that Grounded Theory is often associated with theory development (as the name suggests), but as we pointed out earlier, Grounded Theory studies tend to result in substantive theories, rather than formal theories—the latter exhibiting a higher degree of generalizability (Glaser 1978). Several useful resources on computer simulation research are available. Müller and Pfahl (2008) provide a good starting point, and chapter "The Role of Simulation-Based Studies in Software Engineering Research" in this book provide additional details.

5.6 Solution-seeking Research

The recommended sources listed above focus on knowledge-seeking research. For those researchers who seek to conduct solution-seeking research, we suggest the following sources as a good starting point.

Much of the research published in the flagship conference of our field, the International Conference on Software Engineering (ICSE) tends to present solution-seeking research, by means of a new tool, technique, algorithm, or process. Such papers also include an evaluation of the proposed solution using a knowledge-

seeking strategy. Shaw (2003) presented an analysis of all submitted research papers to ICSE 2002, and sought to distill "patterns" of what constitutes good research in software engineering. Wieringa and Heerkens (2006) has discussed methodological soundness of papers within the requirements engineering domain, and has offered a paper classification and evaluation criteria (Wieringa et al. 2006) to help researchers distinguish between different types of research. In later work, Wieringa (2009) linked this more explicitly to Design Science. Hevner et al. (2004) has discussed Design Science in Information Systems research, a field of research that has considerable overlap with software engineering closely. Design Science is also the topic of chapter "The Design Science Paradigm as a Frame for Empirical Software Engineering."

6 Conclusion

Issues to do with research methodology are receiving increased attention in SE research. However, the field suffers from inconsistent use of terminology and the lack of an integrated and holistic framework within which to categorize research strategies. We seek to provide both consistent terminology and a holistic and integrated framework—the ABC framework. While consistency in labeling research methods may remain challenging to achieve, the ABC framework (with origins in the social sciences (McGrath 1984)) offers useful terminology for what we have labeled *research strategies*.

In this chapter, we describe the two modes of software engineering research and position them within the wider context of conducting software engineering research (Fig. 1): knowledge-seeking and solution-seeking research. Whereas the ABC framework positions eight archetypal research strategies that can be used to conduct knowledge-seeking research, we link Design Science to solution-seeking research. Design Science is discussed in detail in chapter "The Design Science Paradigm as a Frame for Empirical Software Engineering" of this book , which is why we do not discuss this further. Design Science complements the ABC framework, as suggested in Fig. 1; hence, we suggest that the 'ABC' is followed by a 'D,' with D for Design Science.

In addition to providing descriptions, metaphors, and references for each research strategy, we offer practical guidelines to help SE researchers select an appropriate research strategy.

The ABC framework is useful in several ways. First, it offers a systematic approach to explore the landscape of knowledge-seeking research, and as such it serves the purpose of a tutorial. Second, the ABC framework can be used to design a research programme as illustrated in this chapter. Third, the ABC framework can also be used in a reflective manner, for example by categorizing studies as part of a systematic literature review—most systematic reviews organize studies by their research method as *claimed* by the studies' authors. However, due to the confusion that exists within the software engineering field (we elaborate on this point

elsewhere (Stol and Fitzgerald 2018)), in many cases studies are mis-characterized, which leads to a misrepresentation of a research area when presented in a systematic review. We hope the re-discovery of McGrath's circumplex and its introduction to the software engineering field helps to address this issue.

Acknowledgements This work was supported, in part, by Science Foundation Ireland grant 15/SIRG/3293 and 13/RC/2094 and cofunded under the European Regional Development Fund through the Southern & Eastern Regional Operational Programme to Lero—the Irish Software Research Centre (http://www.lero.ie).

References

Australian Electoral Commission (2016) Australian electoral commission image library, 2016 federal election. opening the house of representatives ballot papers (election night). https://upload.wikimedia.org/wikipedia/commons/9/93/AEC-Senate-election-night-opening.jpg, distributed under Creative Commons CC BY 3.0, https://creativecommons.org/licenses/by/3.0

Barcomb A (2019) Retaining and managing episodic contributors in free/libre/open source software communities. PhD thesis, University of Limerick

Barcomb A, Kaufmann A, Riehle D, Stol KJ, Fitzgerald B (2019a) Uncovering the periphery: A qualitative survey of episodic volunteering in free/libre and open source software communities. Transactions on Software Engineering in press

Barcomb A, Stol KJ, Riehle D, Fitzgerald B (2019b) Why do episodic volunteers stay in FLOSS communities? In: Proceedings of the 41st International Conference on Software Engineering, ACM, New York, NY, pp 948–959

Barcomb A, Stol K, Fitzgerald B, Riehle D (2020) Managing episodic volunteers in free/libre/open source software communities. IEEE Trans Softw Eng (in press)

Bos N, Sadat Shami N, Olson J, Cheshin A, Nan N (2004) In-group/out-group effects in distributed teams: An experimental simulation. In: Proceedings of the International Conference on Computer-Supported Cooperative Work and Social Computing (CSCW'04), ACM, pp 429–436

Bourgeois L (1979) Toward a method of middle-range theorizing. The Academy of Management Review 4(3):443–447

Dalkey N, Helmer O (1963) An experimental application of the delphi method to the use of experts. Management Science 9(3):458–467

Damschen E, Baker D, Bohrer G, Nathan R, Orrock J, R Turner J, Brudvig L, Haddad N, Levey D, Tewksbury J (2014) How fragmentation and corridors affect wind dynamics and seed dispersal in open habitats. Proceedings of the National Academy of Sciences of the United States of America 111(9):3484–3489, DOI 10.1073/pnas.1308968111

Easterbrook S, Singer J, Storey MA, Damian D (2008) Selecting empirical methods for software engineering research. In: Shull F, Singer J, Sjøberg DI (eds) Guide to Advanced Software Engineering, Springer

Ebert C, Parro C, Suttels R, Kolarczyk H (2001) Better validation in a world-wide development environment. In: Proceedings of the 7th International Software Metrics Symposium (METRICS)

Edwards H, McDonald S, Young M (2009) The repertory grid technique: Its place in empirical software engineering research. Inform Software Tech 51(4):785–798

Fayerollinson (2010) The victorian civil courtroom at the national justice museum. https://commons.wikimedia.org/wiki/File:Victorian_Civil_Courtroom,

_National_Justice_Museum,_June_2010.jpg, distributed under Creative Commons BY-SA 3.0, https://creativecommons.org/licenses/by-sa/3.0)
Fitzgerald B, Stol KJ, O'Sullivan R, O'Brien D (2013) Scaling agile methods to regulated environments: An industry case study. In: Proceedings of the 2013 International Conference on Software Engineering, IEEE Press, pp 863–872
Glaser B (1978) Theoretical Sensitivity. The Sociology Press
Glaser B, Strauss A (1967) The Discovery of Grounded Theory. AldineTransaction
Glass RL, Vessey I, Ramesh V (2002) Research in software engineering: an analysis of the literature. Information and Software Technology 44(8):491–506
Hassan NR (2015) Seeking middle-range theories in information systems research. In: Proceedings of the 36th International Conference on Information Systems
Hevner A, March S, Park J, Ram S (2004) Design science in information systems research. MIS Quarterly 28(1):75–105
Hoda R, Noble J, Marshall S (2013) Self-organizing roles on agile software development teams. IEEE Transactions on Software Engineering 39(3):422–444
Jiang S, McMillan C, Santelices R (2017) Do programmers do change impact analysis in debugging? Empir Software Eng 22(2):631–669
Juristo N, Moreno A (2001) Basics of Software Engineering Experimentation. Springer Science+Business Media, LLC
Kalliamvakou E, Gousios G, Blincoe K, Singer L, German D, Damian D (2016) An in-depth study of the promises and perils of mining github. Empirical Software Engineering 21(5):2035–2071
Kitchenham B, Pfleeger S (2002) Principles of survey research part 2: Designing a survey. ACM Software Engineering Notes 27(1)
Kitchenham B, Pfleeger S, Pickard L, Jones P, Hoaglin D, Emam KE, Rosenberg J (2002) Preliminary guidelines for empirical research in software engineering. IEEE Trans Softw Eng 28(8):721–734
Kitchenham B, Dybå T, Jørgensen M (2004) Evidence-based software engineering. In: Proceedings of the 26th International Conference on Software Engineering, pp 273–281
Kontio J, Bragge J, Lehtola L (2008) The focus group method as an empirical tool in software engineering. In: Guide to Advanced Empirical Software Engineering, Springer
Krafft M, Stol K, Fitgerald B (2016) How do free/open source developers pick their tools? a Delphi study of the Debian project. In: Proceedings of the 38th ACM/IEEE International Conference on Software Engineering (SEIP), pp 232–241
Lee A, Baskerville R (2003) Generalizing generalizability in information systems research. Information Systems Research 14:221–243, DOI 10.1287/isre.14.3.221.16560
Lethbridge T, Sim S, Singer J (2005) Studying software engineers: Data collection techniques for software field studies. Empir Software Eng 10:311–341
Li Y, Yue T, Ali S, Zhang L (2017) Zen-ReqOptimizer: a search-based approach for requirements assignment optimization. Empir Software Eng 22(1):175–234
Linstone H, Turoff M (eds) (2002) The Delphi Method Techniques and Applications. Addison-Wesley
March ST, Smith G (1995) Design and natural science research on information technology. Decision Support Systems 15(4):251–266
maxpixelnet (no date) Creative Commons CC0 1.0 Universal. https://www.maxpixel.net/Nature-Green-Jungle-Animals-Fauna-Forest-3828424
McGrath JE (1981) Dilemmatics: The study of research choices and dilemmas. Am Behav Sci 25(2):179–210
McGrath JE (1984) Groups: Interaction and Performance. Prentice Hall
McGrath JE (1995) Methodology matters: Doing research in the behavioral sciences. In: Baecker R, Grudin J, Buxton W, Greenberg S (eds) Readings in Human Computer Interaction: Toward the Year 2000, 2nd edn, Morgan Kaufmann Publishers, Inc., pp 152–169
Méndez Fernández D, Passoth JH (2019) Empirical software engineering: From discipline to interdiscipline. The Journal of Systems and Software 148:170–179

Merton RK (1968) Social theory and social structure. Free Press
Mockus A, Fielding R, Herbsleb J (2000) A case study of open source software development: the Apache server. In: Proc. International Conf. Software Engineering
Müller M, Pfahl D (2008) Simulation methods. In: Shull F, Singer J, Sjøberg DI (eds) Guide to Advanced Software Engineering, Springer
Niknafs A, Berry D (2017) The impact of domain knowledge on the effectiveness of requirements engineering activities. Empir Software Eng 22(1):80–133
Nisbett R (2005) The Geography of Thought: How Asians and Westerners Think Differently and Why. Nicholas Brealey Publishing
Ralph P (2015) The sensemaking-coevolution-implementation theory of software design. Science of Computer Programming 101:21–41
Ralph P (2018) Toward methodological guidelines for process theories and taxonomies in software engineering. IEEE Transactions on Software Engineering in press
Runeson P, Höst M, Rainer A, Regnell B (2012) Case Study Research in Software Engineering: Guidelines and Examples. Wiley
Runkel PJ, McGrath JE (1972) Research on Human Behavior: A Systematic Guide to Method. Holt, Rinehart and Winston, Inc.
Russo D, Stol K (2019) Soft theory: a pragmatic alternative to conduct quantitative empirical studies. In: Proceedings of the Joint 7th International Workshop on Conducting Empirical Studies in Industry and 6th International Workshop on Software Engineering Research and Industrial Practice, CESSER-IP@ICSE 2019, Montreal, QC, Canada, May 27, 2019, pp 30–33
Seaman CB (1999) Qualitative methods in empirical studies of software engineering. IEEE Transactions on Software Engineering 24(4):557–572
Setamanit SO (2007) A software process simulation model of global software development (GSD) projects. PhD thesis, Portland State University
Setamanit SO, Wakeland W, Raffo D (2007) Using simulation to evaluate global software development task allocation strategies. Softw Process Improve Pract 12:491–503
Sharma G, Stol KJ (2020) Exploring onboarding success, organizational fit, and turnover intention of software professionals. Journal of Systems and Software 159(110442)
Sharp H, Dittrich Y, de Souza C (2016) The role of ethnographicstudies in empirical software engineering. IEEE Trans Softw Eng 42(8):786–804
Shaw M (2003) Writing good software engineering research papers. In: Proc. 25th International Conf. Software Engineering, pp 726–736
Sim S, Easterbrook S, Holt R (2003) Using benchmarking to advance research: A challenge to software engineering. In: Proc. 25th International Conference on Software Engineering, IEEE Computer Society
Simon H (1996) The Sciences of the Artificial, 3rd edn. MIT Press
Singer J, Storey MA, Sim SE (2000) Beg, borrow, or steal: Using multidisciplinary approaches in empirical software engineering research. In: Proceedings of the International Conference on Software Engineering
Spinellis D, Avgeriou P (2019) Evolution of the unix system architecture: An exploratory case study. IEEE Transactions on Software Engineering in press
Stol K, Fitzgerald B (2015) Theory-oriented software engineering. Science of Computer Programming 101:79–98
Stol K, Fitzgerald B (2018) The ABC of software engineering research. ACM Transactions on Software Engineering and Methodology 27(3)
Stol K, Goedicke M, Jacobson I (2016a) Introduction to the special section—general theories of software engineering: New advances and implications for research. Information and Software Technology 70:176–180
Stol K, Ralph P, Fitzgerald B (2016b) Grounded theory in software engineering research: A critical review and guidelines. In: Proceedings of the 38th International Conference on Software Engineering, ACM, pp 120–131

Stol K, Caglayan B, Fitzgerald B (2017) Competition-based crowdsourcing software development: A multi-method study from a customer perspective. IEEE Transactions on Software Engineering 45(3):237–260

Storey MD, Zagalsky A, Filho FMF, Singer L, Germán DM (2017) How social and communication channels shape and challenge a participatory culture in software development. IEEE Trans Software Eng 43(2):185–204, DOI 10.1109/TSE.2016.2584053

SuperJet International (2011) Full flight simulator. https://www.flickr.com/photos/superjetinternational/5573438825, distributed under Creative Commons CC BY 2.0, https://creativecommons.org/licenses/by-sa/2.0/legalcode

Tyndall J (1896) Fragment of science, volume one. Taken from an electronic copy of the book at Archive.Org (1896 edition of the book) and subsequently annotated in colored typeface. Public Domain, https://commons.wikimedia.org/w/index.php?curid=57653822

Wieringa R (2009) Design science as nested problem solving. In: Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology, DESRIST '09, ACM

Wieringa R, Heerkens M (2006) The methodological soundness of requirements engineering papers: a conceptual framework and two case studies. Requir Eng 11:295–307

Wieringa R, Maiden N, Mead N, Rolland C (2006) Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. Requir Eng 11:102–107

Wohlin C, Runeson P, Höst M, Ohlsson M, Regnell B, Wesslén A (2012) Experimentation in Software Engineering, 2nd edn. Springer

Wohlin C, Smite D, Moe NB (2015) A general theory of software engineering: Balancing human, social and organizational capitals. The Journal of Systems and Software 109

Yin R (2014) Case Study Research Design and Methods, 5th edn. CA: Sage Publications Inc.