This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Please cite as:

Klaas-Jan Stol and Brian Fitzgerald (in press) The ABC of Software Engineering Research. ACM Transactions on Software Engineering and Methodology. DOI: 10.1145/3241743

The ABC of Software Engineering Research

KLAAS-JAN STOL, University College Cork and Lero—the Irish Software Research Centre, Ireland
BRIAN FITZGERALD, University of Limerick and Lero—the Irish Software Research Centre, Ireland

A variety of research methods and techniques are available to SE researchers, and while several overviews exist, there is neither consistency in the research methods covered nor in the terminology used. Furthermore, research is sometimes critically reviewed for characteristics inherent to the methods. We adopt a taxonomy from the social sciences, termed here the ABC framework for SE research, which offers a holistic view of eight archetypal research strategies. ABC refers to the research goal which strives for generalizability over Actors (A), precise measurement of their Behavior (B), in a realistic Context (C). The ABC framework uses two dimensions widely considered to be key in research design: the level of obtrusiveness of the research, and generalizability of research findings. We discuss metaphors for each strategy and their inherent limitations and potential strengths. We illustrate these research strategies in two key SE domains: global software engineering and requirements engineering, and apply the framework on a sample of 75 articles. Finally, we discuss six ways in which the framework can advance SE research.

CCS Concepts: • **General and reference** → *Surveys and overviews*; *General literature*; *Empirical studies*;

Additional Key Words and Phrases: Research methodology, research strategy

ACM Reference Format:
Klaas-Jan Stol and Brian Fitzgerald. 2018. The ABC of Software Engineering Research. *ACM Trans. Softw. Eng. Methodol.* 1, 1, Article 1 (January 2018), 51 pages.
https://doi.org/10.1145/3241743

1 INTRODUCTION

The proper place to study elephants
is the jungle, not the zoo.[1]

The proper place to study bacteria
is the laboratory, not the jungle.[2]

[1]Ephraim R. McLean, comment on a paper by Richard van Horn [135]
[2]Remark by Keng-Leng Siau at a conference.

This work was supported, in part, by Science Foundation Ireland grant 15/SIRG/3293 and 13/RC/2094 and co-funded under the European Regional Development Fund through the Southern & Eastern Regional Operational Programme to Lero—the Irish Software Research Centre (www.lero.ie).
Authors' addresses: K. Stol, School of Computer Science & Information Technology, University College Cork, Ireland; B. Fitzgerald, Lero—the Irish Software Research Centre, University of Limerick, Ireland.

1049-331X/2018/1-ART1
https://doi.org/10.1145/3241743

The choice of research strategy is not 'good' or 'bad' a priori, but is very much dependent on the setting and *goal* of the research. A biologist may be interested in studying the behavior of elephants in groups, for example, in which case a visit to the jungle is warranted, with a considerable lack of control of variables (and the elephant's behavior) as an inherent consequence. On the other hand, a researcher may be interested in studying *parts* of an elephant, such as the structure of its skin, in which the continuous and immediate access that a zoo offers is more appropriate. Besides ease of access, the zoo environment offers a potentially higher degree of control of measurement of the variables that a researcher might be interested in. However, this comes at the inherent cost of a less realistic context.

Research methodology has received considerable attention in the software engineering research community in recent years [48, 84, 155, 190]. In a review published in 2002, Glass et al. concluded that SE research was *"narrow regarding research approach and method"* [63], but this has since changed dramatically. Software engineering researchers have adopted numerous research methods, approaches, and techniques from other fields, partly driven by calls for more evidence-based practice and empirical research. Traditionally, empirical research in software engineering has been characterized by a strong emphasis on quantitative and experimental research [12, 102, 155, 206]. However, more recently, a broader range of approaches have been applied, including qualitative approaches such as grounded theory studies [202], ethnographies [179], and Delphi studies [105].

A key issue in discussing research methods in software engineering is disagreement and confusion about terminology. Two decades ago, Harrison et al. called for *"a classification scheme of research methodologies"* [68] to improve the state of empirical research in software engineering. Thus far, a common taxonomy has been lacking, despite numerous overviews of research methods in software engineering. Table 1 presents a small selection of sources from the SE literature presenting overviews of research methods.[3] These overviews provide a good

[3]Many more sources exist. We selected these because they are well known within the SE literature judging by their citation counts.

Table 1. A "mixed bag": alternative research methods in software engineering according to a selection of sources

Glass et al. [63]	Zannier et al. [230]	Sjøberg et al. [190]	Höfer & Tichy [75]	Easterbrook et al. [48]
Action research	Controlled experiment	Controlled experiment	Case study	Experimentation
Conceptual analysis	Quasi experiment	Surveys	Correlational study	Case study
Concept implementation	Case Study	Case studies	Ethnography	Survey
Case study	Exploratory case study	Action research	Ex post facto study	Ethnography
Data analysis	Experience report		Experiment	Action research
Discourse analysis	Meta-analysis		Meta analysis	
Ethnography	Example application		Phenomenology	
Field experiment	Survey		Survey	
Field study	Discussion			
Grounded theory				
Hermeneutics				
Instrument development				
Laboratory experiment (human / software)				
Literature review				
Meta-analysis				
Mathematical proof				
Protocol analysis				
Phenomenology				
Simulation				
Descriptive/expl. survey				

introduction, but tend to list a "mixed bag" [132] of methods. Typically, such sources enumerate a set of methods, without a holistic view that affords a systematic comparison. Furthermore, when considering the research process as a series of steps, the 'scope' or 'granularity' of various methods and techniques varies widely, ranging from data analysis techniques (e.g. discourse analysis and social network analysis), to a complete research process that addresses data collection, sampling, and analysis (e.g. grounded theory [60]). Terminology can also be ambiguous: terms such as "case study" and "experiment" are sometimes misinterpreted [48, 191]. This ambiguity and lack of a systematic comparison makes it difficult, especially for novice researchers who have embarked on a doctoral program, to understand the trade-offs of choosing one method over another.

Rather than discussing research *methods*, we *"borrow"* (following Sim et al. [187]) the term 'research strategy' [170]. A research strategy is a category of research methods that can be characterized by two dimensions: (a) the level of *obtrusiveness* of the research in a given setting, and (b) the extent to which the findings are *generalizable* to other settings. We discuss these dimensions in more detail in Sections 2 and 3.

All research strategies are inherently limited in one way or another. For example, case studies are often unfairly criticized for their lack of generalizability. Likewise, readers may lament the lack of realism in a controlled experiment conducted in a laboratory setting. Researchers have argued for more realism in software engineering experiments [188], specifically with respect to participants, or *actors* (e.g. students v. professionals), tasks, or what we could also describe as *behavior* (e.g. artificial problems v. real-life problems) and environments, or *contexts* (e.g. pen and paper v. an industrial development environment). Others may find the lack of contextual information in sample survey research unsatisfactory. These *inherent* limitations can never be overcome, no matter how hard a researcher might try. McGrath observed that optimizing a study to achieve generalizability over *actors* (A), precise measurement of their *behavior* (B), in a realistic *context* (C) is impossible, and is a *"three-horned dilemma [since] there is no way—in principle—to maximize all three (conflicting) desiderata of the research strategy domain"* [132]. Consequently, if an evaluation of a study is to be fair, it must be based on how well it achieves its potential strengths, rather than on criteria that it can never fulfill.

Much software engineering research is concerned with the same three components (see Table 2):

- actors (A) which includes software professionals [103, 216], software systems [23], and their users;
- their behavior (B) [33, 218] (e.g. coordination among developers [107]; developer productivity [64] and antecedents e.g. motivation [14]; systems' performance and other quality attributes);
- and the context (C) of a specific system or organization [47, 156, 157].

In this article,[4] we present the ABC framework (named after the three concerns mentioned above) which is adapted from the taxonomy developed by McGrath and his colleagues in the social sciences [130–132, 134, 170], and seek to provide guidance to help researchers select an appropriate research strategy that aligns with the goals of their research. The ABC framework contributes to the discourse on research methodology in software engineering by offering an alternative, holistic view that positions eight archetypal research strategies along the two dimensions of obtrusiveness and generalizability mentioned above.

[4]This article is a revised and extended version of our paper *"A Holistic Overview of Software Engineering Research Strategies"* [199].

Table 2. Examples of Actors, Behavior and Context in software engineering research

Actors	Managers, software engineers, users, software systems, software development artifacts incl. defects, tools, techniques, prototypes
Behavior	System behavior (e.g., reliability, performance and other quality attributes), software engineers' behavior and antecedents such as productivity, motivation, and intention
Context	Industrial settings, organizations, software projects, development teams, software laboratory, classroom, meeting rooms

Table 3. Comparison of knowledge-seeking vs. solution-seeking research in software engineering research

	Knowledge-seeking research	Solution-seeking research
Goal	To generate or propose scientific claims, and to evaluate and validate those claims. This may also include the development of instruments or other artifacts with the specific purpose of supporting or enabling these knowledge-seeking activities, e.g., the construction of an instrument that facilitates data gathering or analysis.	To design or develop new, or improve existing solutions that can help to overcome or ameliorate challenges, bottlenecks, and other problems in the development of software systems and supporting processes.
Focus of research	A phenomenon within software engineering, or a characteristic thereof, that is not (or not sufficiently) well understood or known.	A specific software engineering challenge, obstacle, or problem, and the design or creation of a solution.
Outcome	Empirical findings, descriptions, insights generated by simulations, theoretical or conceptual frameworks, or hypotheses.	Artifacts, which include algorithms, tools, notations (incl. languages), models, mechanisms and techniques.
Example	Research question: How does Extreme Programming work? Sharp et al. conducted an ethnographic study to capture how developers employ XP development practices [180].	Research question: how to select a manageable subset of the input data faster in order to automatically find performance bottlenecks? Luo et al. proposed FOREPOST: an adaptive, feedback-directed learning testing system that helps to find performance bottlenecks [124]. (Note that the *evaluation* of FOREPOST is a knowledge-seeking study.)

We distinguish between *solution-seeking* and *knowledge-seeking* studies [201] (see Table 3 for a summary). *Solution-seeking* studies aim to solve practical problems for which solutions can be engineered. Wieringa has called this type of research *world problems* [220] and later, *practical problems* [219]. In solution-seeking studies, researchers design, create, or develop solutions for a software engineering challenge. The outcome of these studies include algorithms, models, and tools.

Knowledge-seeking studies, on the other hand, aim to learn something about the world around us—and in a software engineering context, that world includes software systems, artifacts resulting from the software development process (e.g. defects), and users and developers along with their behavior, within a given context. Knowledge-seeking studies can also be conducted to evaluate or validate solutions developed in solution-seeking studies, or to compare different solutions for performance, for example. Knowledge-seeking studies are not limited to *empirical* methods only; non-empirical (or theoretical) approaches such as computer simulation and the development of conceptual models also address knowledge questions. In this article we focus on knowledge-seeking studies exclusively. Furthermore, we focus on strategies to conduct *primary* research, and therefore do not consider strategies to conduct *secondary* studies including systematic literature reviews, meta-analyses, and meta-ethnographies. Secondary studies are typically conducted as "desk research" and aim to synthesize or summarize research results presented in primary studies.

This article proceeds as follows. Sec. 2 discusses a number of problems related to terminology in SE research methodology. This section also provides an overview of methodological guidance for SE researchers, and concludes with a discussion of two key dimensions (obtrusiveness and generalizability) that are important in the selection of a research strategy. In Sec. 3, we present the ABC framework which defines eight archetypal research strategies for software engineering research. Sec. 4 demonstrates the applicability of the framework by illustrating how the eight archetypal research strategies are employed in two key research areas within SE: global software engineering (GSE) and requirements engineering (RE). Sec. 5 concludes the article by discussing implications for SE research and how to further advance SE research.

2 BACKGROUND AND RELATED WORK

This section starts with a brief discussion of terminology concerning research methods (Sec. 2.1). We then summarize prior work which offers guidance in selecting research methods (Sec. 2.2). We conclude this section by discussing the two previously mentioned dimensions which are central when choosing research strategies: the level of obtrusiveness of a study, and the generalizability of a study's findings (Sec. 2.3).

2.1 Terminology

Since Glass and colleagues noted the "narrow" range of research approaches [61, 63], the SE community has seen an increased interest in the social and human aspects of software engineering, though the importance of these aspects was already observed much earlier [35, 114, 179]. Traditionally, empirical research in software engineering implied quantitative approaches and experimentation [11],[109, p. 98],[20, 32, 139, 154, 155, 175, 207]. However, there has been an increasing awareness that software engineering is multi-disciplinary [68], a *"social activity"* [42], and *"essentially a human activity"* [175, 222], and that SE researchers must make observations in the *"real world"* [139, p. 17]. Consequently, the SE field has now widely embraced these alternative approaches to study human aspects. A landmark paper in this respect was Seaman's 1999 article on qualitative methods for software engineering [174], which was published in a special issue of *IEEE Transactions on Software Engineering* on empirical software engineering [84]. Besides special issues in key journals in the field [43, 46], there are dedicated events that focus on these topics, most notably the CHASE (Cooperative and Human Aspects of Software Engineering) workshops [40].

While there is now a considerable body of knowledge on research methods and techniques as they pertain to a software engineering context (see Table 1), the SE literature lacks a commonly adopted typology or taxonomy of research strategies that systematically positions them in relation to each other. When discussing research design, researchers often cite a number of dichotomies [56, 62, 81, 186, 222]:

- *field* v. *laboratory* research
- *desk* v. *field* research
- *in vitro* v. *in vivo*
- *quantitative* v. *qualitative* research
- *fixed* v. *flexible* research
- *positivist* v. *interpretivist* research
- *positivist* v. *interpretivist* research
- *inductive* v. *deductive* research
- *exploratory* v. *confirmatory* research
- *rigor* v. *relevance*
- *internal* v. *external* validity

These dichotomous distinctions are useful for researchers to understand some fundamental research design decisions. For example, it is quite clear that *field* research differs from *laboratory* research in a variety of ways, most notably the level of control that a researcher may exert on the research setting. While these distinctions provide some hints as to a researcher's intention, they do not fully convey the details of the research strategy that the researcher may have in mind. Furthermore, these distinctions do not paint the complete picture, and indeed some represent false dichotomies—that is, the dichotomy suggests two mutually exclusive "extremes," as if it were a trade-off, while in actual fact reality is more complicated.

Terminology is often misused when discussing research methods [143, 232]. We illustrate terminological disagreements with a number of examples. Edmondson and McManus defined field studies as *"systematic studies that rely on the collection of original data—qualitative or quantitative—in real organizations"* [50, p. 1155]. Common methods used in field studies are the case study, interview study, or online questionnaires. However, even these terms are problematic. In a review of methods for evaluation, Zelkowitz [231] observed that *"many of the authors used terms like 'experiment,' 'case study,' 'simulation,' 'controlled,' etc. in very different ways."*

The term 'case study' has been particularly problematic. Easterbrook et al. [48] pointed out that, *"There is much confusion in the SE literature over what constitutes a case study."* Indeed, as Easterbrook et al. and others [90] have

observed, the term case study has been used as an empirical method to study a phenomenon (e.g., a case study of open source software development [142]) or as a 'worked example' (e.g., a simulation of adaptive security in a building [211]). Rosenblum and Weyuker [166] used the term "case study" for their study on a regression test suite, but Lanubile considered this an experiment instead, drawing a clear distinction between a study of the artificial and real world events [109, p. 105]:

> *"Although the authors use 31 real versions of the popular KornShell command processor, the test suite is artificial and thus the study is a simulation of the real evolution. The artificiality of the study does not fit with the definition of case study, which focuses on real events."*

Van Horn differentiated between *case studies* and *field studies*, with case studies focusing on a single organization (or part thereof) and field studies considering *"several or more"* organizations [212]. In his description, both methods lack experimental control. Furthermore, he acknowledged that field studies are similar to case studies.

Limiting a study's characterization as a 'case study' or an 'experiment' does not clearly convey the *goal* of the research. Case studies, for example, can be descriptive, exploratory or evaluative, and within each of those, the level of control that a researcher (believes he or she) can exert varies. A descriptive case study typically depends on "thick description" to capture information about a phenomenon within a given real-world context. Some exploratory case studies only present qualitative findings (e.g., Herbsleb and Grinter's study of distributed development [73]), whereas others develop and quantitatively evaluate a set of hypotheses (e.g., the two case studies of open source development by Mockus et al. [142]). The term 'experiment' has been used for several types of research design [191], and Montesi and Lago found that the terms 'experiment' and 'experimentation' are often inappropriately used in software engineering research [143].

The term 'survey' is equally problematic; in most cases it refers to a *sample survey*, but the term has also been used as a synonym for *literature review* (e.g., *"A Survey of Controlled Experiments in Software Engineering"* [191]), or as an overview of existing solutions in a given domain (e.g., *"A Survey on Software Architecture Analysis Methods"* [44]). Other terms have been misused as well; for example, many software engineering researchers claim to have done a "Grounded Theory" study when in reality they may have merely used specific data analysis techniques (e.g., open coding) [202]. Consequently, it is not always clear what is meant by research method, data collection method, or technique. Table 1 presents a selection of mixed bags of methods in the SE literature by different authors—several other overviews exist; the aim of this table is not to be exhaustive, but rather to illustrate the point that different authors have presented quite varying overviews of methods. Some sources provide more detailed classifications than others. For example, some sources use the term experimentation, whereas others differentiate between controlled experiment and quasi experiment.

It is important that studies report clearly which research strategy has been employed in order to prevent misunderstandings. One example of this arose around an experiment by Sobel and Clarkson [192]. This experiment investigated the effects of teaching formal methods to undergraduate software engineering students. In a comment on this study, Berry and Tichy argued that the experiment was invalid for several reasons, including weaknesses in the experimental design and a lack of control [17]. In turn, Sobel and Clarkson issued a rebuttal claiming that their experiment was not *laboratory* research, but rather *field* research and claimed that, as a consequence, Berry and Tichy's points of critique were moot [193].

Defining a common taxonomy of methods is very complicated for several reasons. One difficulty is that research methods tend to be of different 'granularity.' A case study, for example, implies quite a specific scope of study (which is what makes it a *case* study), and researchers typically define a unit of analysis [229]. However, the actual data collection and analysis methods are not prescribed—while common data collection methods include interviews, case studies could also rely on quantitative data only. An ethnographic study is more precisely focused as being conducted within a very specific and natural context, and data collection is done in unobtrusive ways—that is, the researcher does not manipulate or control the research setting, but merely aims to understand

it. On the other hand, the scope in Grounded Theory studies is not readily clear. GT studies can be conducted within a specific organization (e.g. [1]) or across organizations (e.g. [74]), depending on the goal of the study.

A second complication is that hundreds of methods and techniques exist, and new methods are proposed as researchers become dissatisfied with existing approaches or develop new techniques based on technological advances. Grounded Theory, for example, was developed in the 1960s as a result of a dissatisfaction with how contemporary research was conducted [59]. Newly developed theories also give rise to new methods. For example, *Personal Construct Theory* (PCT) gave rise to an approach called the Repertory Grid Technique [51]. Social Network Analysis (SNA) is another well-known data analysis technique that can be traced back to sociology, anthropology, and role theory [205]. Today, SNA has been used to study software development activity and developers [123], which is enabled by the rich availability of data and technology to analyze these networks.

2.2 Related Work

The selection and appropriate use of research methods and strategies is critically important to conduct sound research. Numerous authors have provided guidance and analyses towards this goal—Table 4 lists a number of well known sources. Similar to Table 1, the purpose of Table 4 is not to be exhaustive, but rather to list some of the key sources that are available to SE researchers, and to indicate the range of variety in different methods that have been used in SE research. Several of the sources in the table provide an analysis on the use of, or guidance for, specific methods. For example, extensive guidance is available for conducting case studies [168, 169], experiments [91, 223], and survey studies [158]. Other sources discuss the use and reflect on the role of approaches such as the Repertory Grid Technique [51], Grounded Theory [202], and ethnographic studies in software engineering [179].

Seaman presented one of the first in-depth overviews of qualitative methods for software engineering research, in particular focusing on data collection methods and analysis techniques such as interviews and the constant comparison method found in Grounded Theory [174]. This article represented a major milestone in SE research, given the hitherto strong focus on quantitative methods. Lethbridge et al. proposed a taxonomy of data collection techniques which is specifically focused on field studies [117]. Their taxonomy is organized around the level of human intervention; that is, how much *involvement* of the researcher or participants is needed to collect data. Techniques vary from first degree techniques such as interviews, second degree such as instrumenting systems and fly-on-the-wall, to third degree techniques such as document analysis.

A number of authors have presented typologies and taxonomies of research approaches in software engineering. Shaw developed a bottom-up classification of research questions and research methods based on an analysis of paper abstracts submitted to the 2002 *International Conference on Software Engineering* (ICSE) [183]. Her classification addresses questions regarding generalizability or characterization of phenomena, methods for analysis, and design and evaluation of an artifact or practice. Wieringa et al. [221] proposed a paper classification based on an "engineering cycle," which focuses specifically on *Requirements Engineering* and consists of activities such as problem investigation, solution design, and solution validation. The paper classification distinguishes the following types of research: evaluation research, proposal of solution, validation research, philosophical papers (which include conceptual papers), opinion papers, and personal experience papers. Though the classification was suggested specifically for the RE field, it has proven useful in many other areas within software engineering, in particular for classifying studies in systematic reviews and mapping studies (e.g. [157]). Montesi and Lago proposed a taxonomy of SE article types based on previous literature discussing research methodology, author instructions provided by selected journals, and calls for papers of major SE conferences [143]. While these classifications are useful, they does not assist in understanding the inherent strengths and limitations of the various research strategies that a researcher may select to conduct research.

Table 4. Selection of methodological guidance relevant to software engineering research in the last two decades

Authors	Year	Focus	Contribution
Zelkowitz and Wallace [232]	1998	Technology validation	Provides an overview of experimental techniques to validate of new technologies.
Seaman [174]	1999	Qualitative methods	Guidelines for qualitative data collection and analysis.
Wohlin et al. [223, 224]	2000, 2012	Experiments	Book with a brief overview of empirical methods in SE with the remainder focused on conducting experiments. Second edition published in 2012.
Juristo and Moreno [91]	2001	Experiments	Book on experimentation in software engineering.
Pfleeger and Kitchenham [97–101, 158]	2001-2003	Surveys	A six-part tutorial series on designing and conducting surveys in software engineering.
Kitchenham et al. [102]	2002	Experimental studies	Guidelines for conducting (experimental) empirical studies.
Shaw [182, 183]	2002	Software engineering studies in ICSE 2002	Overview of types of research questions, research results, validation techniques, and research strategies based on analysis of ICSE 2002 papers.
Lethbridge et al. [117]	2005	Data collection methods for field research	A typology of data collection methods based on the degree of human intervention. Distinction between first, second, and third degree methods.
Wieringa et al. [221]	2006	Requirements engineering research	Framework for classifying RE research: evaluation, validation, opinion, solution proposal, philosophical, personal experience.
Easterbrook et al. [48]	2008	Empirical software engineering research	Overview of common methods. Selection of methods informed by type of research question and epistemology. (Chapter in Shull et al. [185] below).
Shull et al. [185]	2008	Empirical software engineering	Collection of chapters that discuss variety of research methods, including focus groups, simulation, experiments and theory development.
Runeson et al. [168, 169]	2009, 2012	Case studies	Guidance for designing, conducting and reporting case studies; the 2009 article was extended into a book published in 2012.
Edwards et al. [51]	2009	Repertory Grid Technique	Review and discussion of the Repertory Grid Technique in SE.
Ivarsson and Gorschek [79]	2011	Rigor and relevance	Presents a model for evaluating rigor and relevance of technology evaluations for industry.
Wohlin and Aurum [222]	2015	Empirical software engineering research	A decision-making structure for selecting a research design, considering issues such as research question, research logic, and research purpose.
Sharp et al. [179]	2016	Ethnographic studies	Discusses the use and value of ethnographic studies in SE research.
Stol et al. [202]	2016	Grounded theory	Discusses different variants of grounded theory, GT use in SE studies, and guidelines for reporting GT.
Ralph [161]	2018	Developing process theories and taxonomies	Guidance for developing process theories (as opposed to variance theories) in software engineering.

Wohlin and Aurum proposed a decision-making structure for selecting a research design [222]. Their framework considers a number research design decisions including the research outcome, research logic (i.e. inductive vs. deductive), and research purpose (i.e. explanatory, descriptive, exploratory, or evaluative).

Most authors discuss *empirical* research methods, which refers to research approaches to gather observations and evidence from the real world. However, knowledge-seeking studies may also adopt non-empirical, or theoretical, research strategies which can provide useful insights, such as formal theory development [161, 189, 200] and simulation [77]. While the various classifications discussed above each have a specific focus and strengths, their discussion remains at the level of individual research methods and techniques. Furthermore, they

do not provide a holistic overview of research approaches that systematically positions and compares alternative strategies in relation to one another. In the next subsection, we lay the foundation for such a holistic overview.

2.3 Dimensions of Research Strategies: Obtrusiveness and Generalizability

Two important dimensions of research strategy are the level of *obtrusiveness* and *generalizability*. These two dimensions are the axes in the ABC framework in Sec. 3, and are described in turn here.

2.3.1 Obtrusiveness. The first dimension is concerned with how *obtrusive* the research is: to which extent does a researcher "intrude" the research setting, or simply make observations in an unobtrusive way. Researchers who wish to exert more control in a research study usually have to intrude in the research setting, for example to manipulate some variables. Several authors have argued that the level of control that a researcher can exert while conducting a study is a key concern [35, 54, 67, 154, 169, 232]. One concern often raised in the context of qualitative methods such as ethnographic and case study research is a lack of control. Parnas lamented the lack of experimental design in current empirical software development research and how observations from a small number of 'uncontrolled case studies' do not contribute to scientific knowledge [154, p. 56]. In response to Parnas, Curtis argued that *"the more realistic the experimental environment, the more difficult it becomes to control all the factors that create alternate explanations of the hypothesized results"* [34]. This clearly suggests the inherent tension between the experimental environment (Context) and precision of measurement (of Behavior). Expectations as to what represents a "good case study" also vary. For example, some reviewers have lamented the *"[lack of] control usually required to do a good case study paper,"* or that studies are *"poorly controlled"* [35, p. 1099]. However, in many cases, the purpose of an exploratory or descriptive case study is to *understand* the challenges and practices in a real-world setting, rather than *measuring* any relationship between variables. Thus, the level of control that a researcher has (or thinks to have) will depend on the goal of the research and will affect the realism of the research context. Hannay and Jørgensen distinguished between the 'structural' and 'situational' artificiality in experiments: structural artificiality is *"the methodological essence of control"* to allow for controlling of variables and drawing conclusions about treatment-outcome relationships [67]. Situational artificiality refers to the elements of the experimental design, such as the subjects (e.g., the use of students) and tasks and settings (e.g., toy systems). Selecting a research approach with high potential for control does not guarantee that this is achieved in practice. For example, a controlled experiment offers considerable *potential* for a high level of control, but this can be very difficult to achieve and such a study requires a careful design (e.g., [17, p. 569]).

2.3.2 Generalizability. A second key concern that authors have expressed is the level of generalizability of research findings. This has been a recurring concern in software engineering research, in particular in the context of case studies— captured succinctly by one referee: *"Case study reports are [...] limited, because they report a single case."* Indeed, exploratory case studies, and other types of field studies, are limited in that the researcher cannot draw any statistically generalizing conclusions from such studies. However, such 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. For example, Mockus et al. developed hypotheses based on a case study of one open source project and tested these through a second case study [142].

Ethnography is another research method that has been adopted by software engineering researchers [164]. Van Maanen defines the aim of an ethnography as: *"to discover and disclose the socially acquired and shared understandings necessary to become a member of a specified social unit,"* and the result to be a *"cultural description"* [213, p. 103]. Thus, the scope of the setting is by definition limited. This inherent limitation of the method should not be highlighted as a shortcoming when evaluating such a study.

3 THE ABC FRAMEWORK FOR RESEARCH STRATEGIES

This section presents a framework that provides a holistic overview of eight different research strategies. The framework, which we have termed the ABC framework (Fig. 1), was originally devised by McGrath and his colleagues for the social sciences [131, 133, 134, 170, 217]. We have interpreted, tailored, and operationalized the framework for a software engineering research context. Specifically, we refined the terminology used in the framework—for example, in the social sciences the term "actor" refers to a person, whereas in SE, actors may represent professionals, applications and systems, or their users. To operationalize the framework, we identified exemplar studies that adopted the archetypal research strategies in two SE research areas (global software engineering and requirements engineering). To demonstrate a more general applicability of the framework, we analyzed all articles published in Springer's *Empirical Software Engineering* journal in 2017. Both analyses are presented in Sec. 4.

The framework is underpinned by the two key dimensions of *obtrusiveness* and *generalizability* which were described in Sec. 2.3. These axes frame an area within which eight archetypal research strategies can be positioned. The first dimension (the x-axis in Fig. 1) is that of generalizability or universality: research strategies can result in findings that are either *specific* to a particular context or system, or more *generalizable*. The second dimension (the y-axis in Fig. 1) is the level of 'obtrusiveness,' and refers to the degree to which a research setting is manipulated or instrumented to conduct research. A researcher may be unobtrusive by merely observing an activity or interviewing informants without manipulating any aspect of the research setting—an industry case study using document analysis and interviews to gather data is one example of this. On the other hand, a researcher may 'intrude' on the research setting by manipulating some variables, or divide participants into different treatment groups in order to measure some effect; such operations change the circumstances of a study and its participants.

An article may report on one or more studies, each of which may employ a different research strategy; for example, a case study followed by a sample study [197]. As discussed in Sec. 1, many SE studies are solution-seeking studies, which result in an artifact such as a new algorithm, approach, or tool. Such solution-seeking studies do not fall within the scope of the ABC framework. However, it is customary that such proposed artifacts are evaluated or validated, and such evaluation studies *can* be classified using the ABC framework.

The remainder of this section presents each of the eight research strategies. For each strategy we discuss a metaphor that, in our opinion, conveys the essence of that strategy and which may help in distinguishing the eight strategies. For example, we adopted the "jungle" metaphor from this article's opening quote to represent field studies. Metaphors may help to understand the essence of concepts and terms, but metaphors have limits and should not be taken literally—we do not suggest that field studies (i.e., jungle) pose any danger to researchers. Instead, it suggests that a researcher enters a research setting that existed prior to, and independent of, the researcher's presence. In this setting, we see the researcher as an "explorer" who aims to study or observe a subject in this natural setting without disturbing the actors inhabiting the setting. Because the eight research strategies differ quite significantly in terms of their typical setting and procedures (e.g. a controlled experiment in a contrived setting vs. an ethnography in a natural setting), the metaphors also vary quite significantly. The different metaphors are not meant to be compared, though some are in related domains (e.g. jungle, nature reserve, and greenhouse). Table 5 presents a summary of the research strategies and their associated metaphors, purpose, methods and inherent limitations.

3.1 Field Studies

The term *field study* refers to any research conducted in a specific, real-world setting to study a specific software engineering phenomenon. The field study strategy is located in Quadrant I (Fig. 1), representing *natural settings*. Field studies are unobtrusive in that a researcher does not actively control or change any parameters or variables. That is, there is no deliberate modification of the research setting. Field studies are used to develop a deep

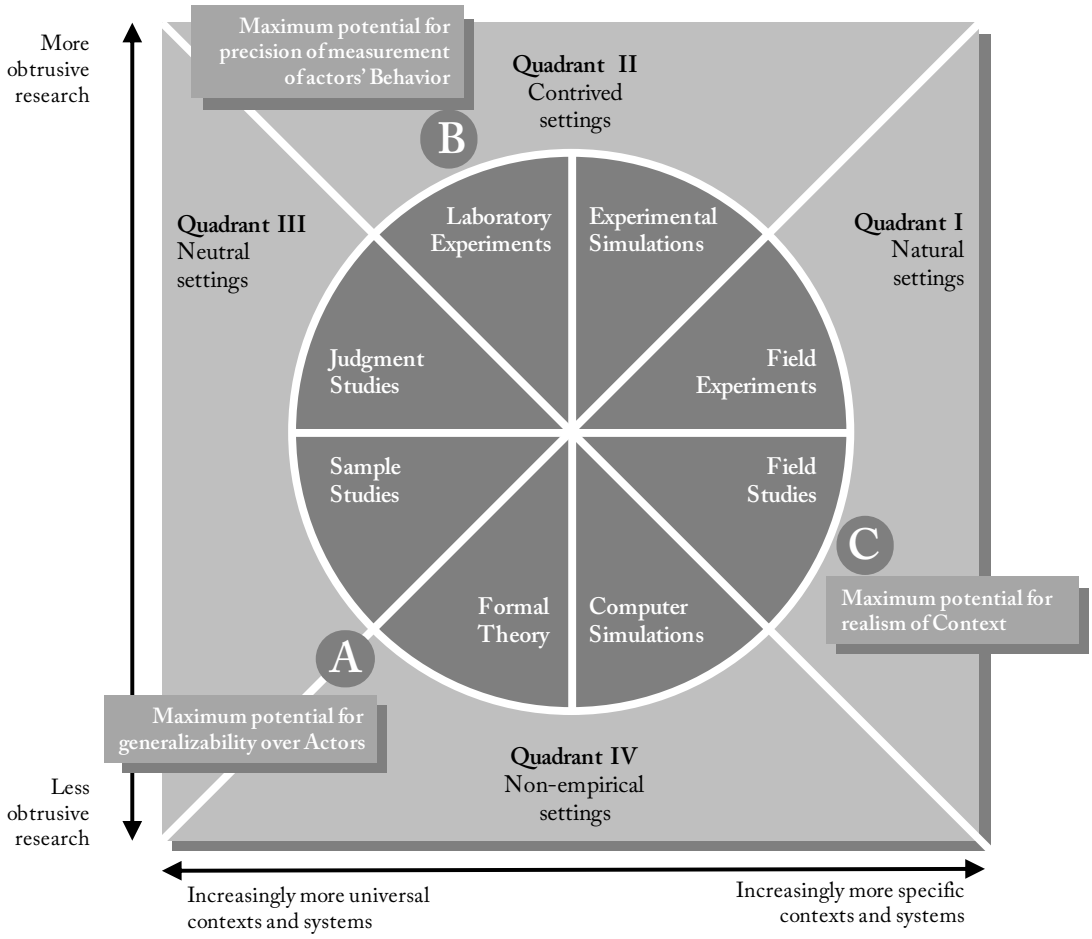


Fig. 1. The ABC framework: eight research strategies as categories of research methods for software engineering (Adapted from Runkel and McGrath [170]). Quadrants I to IV represent different research settings.

understanding of phenomena in specific, concrete, and realistic settings—the specific setting may refer (among others) to a particular system, organization, or team of individuals. For this reason, a field study offers maximum potential for capturing a realistic context (indicated by the 'C' marker in Fig. 1), unlike, for example, a laboratory experiment (see Sec. 3.4). However, this realism is gained at the price of a low precision of measurement of behavior (point 'B' in Fig. 1) and a low generalizability of findings (point 'A').

It is important to note that the mere fact that data is collected in a "field setting" does not imply the study is a field study. For example, a focus group conducted on a company's premises does not automatically make it a field study. Instead, the question is whether or not the field setting is essential to the study, or whether or not the study captures any realistic context. A focus group study could be conducted in any appropriate space. The same applies to studies that mine software repositories.

As mentioned, the metaphor we chose for field studies is the *jungle* from McLean's opening quote (see Sec. 1). A jungle is usually an undisturbed setting, and so wildlife can be observed in its natural habitat—but, the researcher

is expected to not disturb the natural setting as this would lead to different behavior, which in turn reduces the realism of the context. Field studies have become a common strategy within software engineering since the mid-nineties, though some field studies were conducted in the eighties [35]. A common method in SE research that falls within the field study strategy is the descriptive or exploratory case study (e.g. [73, 141, 198]).

One example of a field study in SE is an ethnographic study of XP (Extreme Programming) by Sharp et al. [180]. This study did not aim to 'control' any variables, nor to change anything in the case study setting. Rather, the authors stated that their, *"motivation is to gain insight into the culture and community of an agile method."*

3.2 Field Experiments

A *field experiment* refers to an experimental study conducted in a natural setting with a high degree of realism (similar to a field study), but in this strategy the researcher manipulates some properties or variables in the research setting so as to observe an effect of some kind—this manipulation reduces the level of realism compared to a field study. The realistic research setting exists independent of the researcher, which distinguishes it from a the contrived research setting in a laboratory experiment. Common settings for field experiments in SE include a specific software development organization or team, or a deployed production system.

The term "experimental" should be interpreted in a broad sense, rather than in a strictly scientific sense. In our broader interpretation, *Action Research* [7] also falls within this strategy. In Action Research, a researcher aims to intervene and improve a specific setting through a cycle of making changes, observing the resulting situation, and making further changes. The researcher is "experimenting" by making adjustments and observing the effects of those adjustments. Similarly, the goal in an *in vivo* controlled experiment is to manipulate certain independent variables while measuring some dependent variables. While the level of obtrusiveness is higher than in a field study as a researcher is actively making changes to the research setting, the natural study setting is realistic but subject to confounding factors that limit the precision of measurement (hence, the field experiment is distant from point 'B'), and findings are limited in their transferability to other settings (hence a very long distance to point 'A'). For example, the improvements achieved using Action Research in a particular organization may not easily transfer to other organizations as the researcher's interventions are likely to be dependent on the specific organizational context.

We suggest a *nature reserve* as a metaphor for a field experiment setting. In a nature reserve, flora and fauna can still thrive as normal, but the reserve facilitates the conduct of research, for example by placing fences so as to separate the wildlife into different treatment groups and evaluate the effects of those treatments. An example of a field experiment in SE is a study by Anda et al. in which four companies were selected to implement the same software system [13]. The systems were implemented by developers in real companies; for the developers, the setting was natural, and existed before the researchers entered. The study's goal was to investigate reproducibility of software projects.

3.3 Experimental Simulations

The *experimental simulation* strategy is one of two strategies in Quadrant II—the quadrant that represents *contrived* research settings. In an experimental simulation the behavior of actors (e.g. developers, users, or software systems) that a researcher aims to observe and measure are natural, but the setting within which these occur has been specifically created for the purposes of the study—that is, without the study, the setting would not exist. Thus, the term *simulation* in this context refers to the artificially created setting that aims to recreate a concrete type of setting in which the 'experiment' or observed behavior takes place. The level of obtrusiveness is higher than that of a field experiment, because the simulation requires a *contrived* setting. As the setting is contrived, a researcher is often required to make certain simplifying assumptions—it is often too costly to recreate a highly realistic setting, though the degree of realism can vary considerably across different studies adopting this strategy. In any

case, the context of such a setting is less realistic than that of a field study (where realism is potentially maximized, as indicated by point 'C' in Fig. 1) or a field experiment. On the other hand, the findings from experimental simulations allow a researcher to more precisely measure actors' behavior or systems' properties than would be the case in a field experiment—this strategy lies closer to point 'B' where such precision can be maximized.

We borrow Runkel and McGrath's metaphor, who compared an experimental simulation to a *greenhouse* [170]. A greenhouse is built to simulate a certain setting, optimized for certain characteristics, for example, to grow fruits that require much sunlight and high temperatures, which otherwise could not be grown in cold climates. Compared to a nature reserve (i.e., field experiments), a greenhouse is a far more contrived setting and less realistic, but it gives a researcher potentially more control over what happens inside. An alternative metaphor is a *flight simulator* [170]. While the specific series of events cannot be fully controlled (it depends on user input), the behavior of participants (e.g., pilots in training) can be carefully monitored and analyzed. The flight simulator metaphor is a computer-based virtual environment—Travassos and Barros [210] adopted Tisseau's term *"in virtuo"* [208] for such experiments, to distinguish it from *in vivo* experiments (i.e., field experiments) discussed above. One example of an experimental simulation is a study on design competitions in the context of crowdsourcing software development [110]. The design competitions were artificially created specifically for the study—the context was therefore contrived and not what one would find in a real crowdsourcing setting (as found, for example, in a field study on crowdsourcing [198]).

3.4 Laboratory Experiments

The *laboratory experiment* is the second strategy in Quadrant II. It differs from field experiments which are positioned in Quadrant I and thus study phenomena in their natural context, whereas laboratory experiments are set in a *contrived* setting. A laboratory experiment is characterized by a high potential to neutralize any confounding factors and extraneous conditions [170, p. 105]. Consequently, laboratory experiments allow a researcher to exercise maximum precision of measurement of behavior on the studied object—this is indicated by the 'B' marker in Fig. 1. Clearly, such a level of control would not be possible in a real-world software development environment. However, as a result, the context in a laboratory experiment is very unrealistic—the research setting is contrived and specifically designed for the study at hand. Indeed, this strategy is far removed from the 'C' marker in Fig. 1. Furthermore, laboratory experiments may involve a limited number of subjects (different system setups or human participants—or, as McGrath observed, whomever can be *"lured into the laboratory"*), and consequently the findings of a laboratory experiment are limited in their generalizability. Laboratory experiments differ from experimental simulations in that the former offers a higher degree of precision of measurement. In laboratory experiments, the researcher conducts "discrete" trials of relatively short time span. In experimental simulations, in contrast, a researcher offers an environment where the more continuous flow of events depends on the simulation environment and the actors' behavior.

We do not imply the term laboratory experiment to mean a *controlled* or *quasi experiment* exclusively, but *any* investigation to establish a quantitative *"relationship between several variables or alternatives under examination"* [91, p. 10]. In this interpretation, benchmarking studies that compare a number of algorithms or techniques based on a predefined set of criteria represent "experimentation" too, because the researcher actively sets up a contrived environment to measure and analyze those algorithms and techniques. In SE research, a common approach is to set up a dedicated computer to compare different algorithms or techniques.

A useful metaphor for laboratory experiment is a *test tube*, which Runkel and McGrath [170] used to set it clearly apart from the greenhouse (experimental simulation, representing a more continuous flow of events) and nature reserve (field experiment). This also aligns well with the term "in vitro" (i.e. *in glass*) used to characterize laboratory studies. Alternatively, we could think of it as a *cleanroom*, in which there is a high degree of control over what happens inside. It is worth noting that the term *laboratory* does not imply an actual laboratory, and

that laboratory experiments can be conducted at software development organizations as well. Likewise, not all experiments conducted with students are necessarily laboratory experiments—as per the description above, this depends on the extent to which the experimental setting was contrived. After all, for students, a computer laboratory can pass as a natural setting.

An example of a laboratory experiment is a study on pair programming [5]. For this study, 295 Java consultants with varying levels of seniority were hired to participate. The task at hand was to make changes to two alternative Java systems with different levels of complexity. It is clear that the study setting was contrived; the 295 participants would not have undertaken these tasks without explicitly being requested (and paid). The change tasks themselves were also carefully designed as part of the study.

3.5 Judgment Studies

A *judgment study* involves gathering empirical data from a group of participants who are asked to judge or rate behaviors, to respond to a request, or 'stimulus' offered by a researcher, or to discuss a given topic of interest. Judgment studies rely on *systematic sampling* rather than representative sampling and should involve experts, appropriately informed to respond to a certain question or stimulus. The goal of a judgment study is to seek generalizability over the *responses*, rather than generalizability to a *population* of actors. Using a judgment study a researcher is actively involved to steer the direction and progress of the study. Judgment studies are positioned in Quadrant III, which represents 'neutral' settings—that is, the setting plays no role in the study. In fact, a researcher may actively aim to "mute" the setting so as to ensure that the setting bears no effect on responses. There is no experimental setup or otherwise contrived setting, as is the case in laboratory experiments and experimental simulations. Instead, the researcher aims to "cancel out" the setting from the research design. The researcher is merely interested in the responses of the participants regarding a given question or stimulus, such as opinions and expertise on a subject matter.

One method that fits well with this strategy is the Delphi method which was developed in the late 1940s [36]. The method can be used to structure a group communication process to deal with complex problems [121, p. 3]. A Delphi study comprises a panel of experts, and allows the researcher *"to elicit their input through an iterative, controlled feedback process"* [78, p. 93]. Another method within this strategy is the 'focus group,' which are *"carefully planned discussions, designed to obtain personal perceptions of the group members on a defined area of research interest"* [104, p. 94]. In focus groups, participants are also systematically selected based on their expertise and characteristics. A third and very common instance of judgment studies is the evaluation study whereby participants are typically asked to judge, for example, the utility of an approach or technique.

We liken a judgment study to a *courtroom*, in which a panel of participants (the jury) are carefully and systematically selected. In a courtroom, evidence is presented (a *stimulus*) and eventually the jury returns a verdict. The setting itself (i.e. the courtroom) is only manipulated to the extent that it aims to be neutral and not distracting the participants from the matter at hand (i.e. the case). An example of a judgment study is an investigation of key characteristics for effective tailoring of agile methods [30]. The study was conducted with a panel from both industry and academe that were systematically selected for their expertise.

3.6 Sample Studies

Also in Quadrant III is the *sample study* strategy, which aims to achieve generalizability over a certain population of *actors*, whether these are software professionals, software systems, or artifacts of the development process. The sample study is one of two strategies with the potential to maximize generalizability to a population—this is indicated by the marker 'A' in Fig. 1. The sample study is one of the most common strategies in SE research (see Tables 1 and 8). When the sample consists of human respondents, data is usually collected through questionnaires that can be administered in hard copy or through websites. In SE research, the sample study is also used quite

frequently to study large sets of software development artifacts or projects, in particular open source software projects, which are easy to access. In that case, data collection is usually performed by mining software repositories. A sample study is unobtrusive, in the sense that the researcher does not manipulate any variables during data collection. As a result, the level of precision may be affected—for example, a researcher has no control over respondents to a questionnaire who misunderstand questions. Hence, the sample study strategy is removed from point 'B' in Fig. 1. When collecting data from repositories, again, no variables are manipulated. At best, the researcher can conduct correlational analysis, but no causal relationships can typically be inferred. Positioned in Quadrant III, the research setting is neutral, and consequently, the sample study cannot capture a realistic context (as suggested by the far distance to point 'C'); the goal is to generalize, and this means the focus is not on specific contextual details.

A metaphor for the sample study strategy is a *referendum* (though we admit this only suggests samples of human participants, not development artifacts). In a referendum, usually a limited set of questions is presented to a large group of people, who are invited to respond—typically, only a sample actually responds. An example of a sample study is a survey by Storey et al. [203] who investigated communication channels used by software developers. In particular, their study received a large number of responses (over 1,400), suggesting a representative sample of the GitHub developer population, and thus a high degree of generalizability of the findings. An example of a sample study of software repositories was conducted by Ray et al. [162], who investigated the correlation between programming language and quality. One of their findings was that strong typing is correlated with moderately higher quality code than weak typing. In their study, they could not actively manipulate any variables, and so no causal relationships could be established.

Table 5. Research strategies, their metaphors, purpose, methods and inherent limitations

Study Type	Metaphor and setting	Purpose	Typical methods & data	Inherent limitations
Field Study	*Jungle*: Natural setting that exists before the researcher enters it. Minimal intrusion of the setting so as not to disturb realism, only to facilitate data collection.	Facilitates study of phenomena and actors and their behavior in natural contexts. Exploratory, to understand *'what's going on,' 'how things work,'* or to generate hypotheses.	Case study, ethnography, observational study; qualitative data incl. interviews, field notes, archival documents, may include quantitative data.	• No statistical generalizability • No control over events • Low precision of measurement
Field Experiment	*Nature reserve*: Natural, pre-existing setting (in vivo), but some level of intrusion due to the deliberate manipulation of aspects of the setting; study affected by confounding factors.	To investigate, evaluate, or compare techniques, practices, processes, or approaches within a real-world and pre-existing setting.	Evaluative case study, quasi-experiment, Action Research; studies may use either quantitative data or qualitative data.	• No statistical generalizability • Precision of measurement affected by confounding contextual factors
Experimental Simulation	*Greenhouse, Flight simulator*: Contrived setting (in virtuo) created specifically for a study to represent a concrete type of setting. Environment is created by the researcher to study behavior of actors.	To study behavior of participants or systems in a controlled setting that resembles a real-world, concrete class of settings as close as possible.	Simulation games, management games, instrumented multi-player games; quantitative or qualitative data, depending on the simulation instrument.	• Generalizability reduced as setting is designed to mirror a specific type of setting • Realism reduced due to artificial setting

Continued on next page

Continued from previous page

Study Type	Metaphor and setting	Purpose	Typical methods & data	Inherent limitations
Laboratory Experiment	*Cleanroom, Test tube*: Contrived setting (in vitro) created specifically for a study, with high degree of control of all measured variables.	To study with a high degree of precision relationships between variables, or comparisons between techniques; may allow establishment of causality between variables.	Randomized controlled experiments and quasi-experiments, comparative evaluations benchmark studies; usually quantitative data exclusively.	• Abstract or unrealistic context due to highly artificial setting • Typically scope of problem reduced to study the 'essence,' optimizing internal validity at cost of external validity
Judgment Study	*Courtroom*: Neutral setting; may be actively designed to nullify the context, so that 'responses' are in relation to some stimulus (question or instructions), independent of setting.	To elicit information from subjects for purposes of evaluation or study of an object. To seek generalizability of responses to stimuli, not generalizability to a population.	Delphi studies, interview studies, focus group, evaluation studies; use of qualitative and/or quantitative data.	• Responses not related to any specific or realistic context • Less generalizability than sample studies due to lack of *representative* sampling • Less control and precision of measurement than a lab. exp.
Sample Studies	*Referendum*: Neutral setting. Limited level of precision of measurement; no variables are manipulated. The researcher must deal with whatever data is collected.	To study the distribution of a particular characteristic in a population (of people or systems), or the correlation between two or more characteristics in a population. Information is sought of the subjects.	Software repository mining, questionnaires, interviews; analysis includes correlational methods e.g. regression. Typically, quantitative data (e.g. Likert scales) but can include qualitative data.	• Reductionist—depth of and number of data points per participant limited • Data collection not 'interactive': no option to clarify questions; repository data comes *as-is*, no opportunity to *manipulate* variables, only to *correlate* them
Formal Theory	*Jigsaw puzzle*: Non-empirical setting; typically a research office or library.	To develop a conceptualization, framework or theory on a topic. Focus is on formulating relations among concepts, or explanations that hold for a wide range of contexts.	Conceptual reasoning, concept development, development of propositions and/or hypotheses; framework development.	• Low on realism: does not consider a specific context but rather abstract concepts • No manipulation of variables or measurement (no empirical information is gathered)
Computer Simulation	*Forecasting system*: Non-empirical setting (in silico); no recording of observations in the real world. There are no actors (people, real-world systems) or real-world behavior: everything is specified in the simulation.	To model a particular system or phenomenon that facilitates evaluation of a large number or complex scenarios that are captured in the pre-programmed model.	Development of software programs that contain symbolic representations of all variables a researcher considers important; usually these variables are derived and calibrated based on prior empirical studies.	• No manipulation of variables or precision of measurement (no empirical data is gathered) • Results will be as good as the accuracy of the model representing the simulated system • Low generalizability as it attempts to model a specific class of real-world systems

3.7 Formal Theory

Formal theory is one of two strategies in Quadrant IV, which have *no empirical* setting. Formal theory[5] is a strategy that aims at a high level of universality so that the resulting theory or framework can be applied under

[5]Not to be confused with formal methods that are used, for example, to develop formal program specifications.

a wide array of circumstances, although most theories have boundaries outside of which they do not apply [200]. This maximum potential for generalizability over a population of actors is indicated by the marker 'A' in Fig. 1, which is shared with the sample study strategy. The result of this research strategy is not necessarily a theory in the traditional sense of the word (i.e. a process or variance theory [161, 200]) but can also comprise development of conceptual or research frameworks. Thus, in our use of the term "theory" we include any form of conceptualization or frameworks that seek to have some degree of generalizability. Formal theory and its role in software engineering research has received considerable attention in recent years, including a series of workshops and special issues in journals [189, 201]. Formal theory is a desk research activity, and does not involve any concrete or realistic context (hence, it is distant from point 'C'), nor does it involve any empirical measurement of behavior (it is positioned far from point 'B' also). However, the development of theories and conceptual frameworks typically depends on prior empirical observations.

As a metaphor, we propose that developing formal theory is akin to solving a *jigsaw puzzle*. Solving a jigsaw puzzle can be done as a solitary or team effort, happens in a non-empirical context (e.g., at a large table to fit all the pieces), and the goal is to "fit" all pieces together. In particularly complex jigsaw puzzles, some pieces will be 'theorized' as representing the sky or water (both are variations of blue) and as the puzzle proceeds, more and more pieces fit together. Theorized pieces may have to be validated through empirical studies using other research strategies. As well, the "boundary" of the puzzle may also have to be empirically established using empirical strategies. One example of a study that we classify as formal theory development is the classification and comparison study of architecture description languages (ADL) [136]. The three main elements of an architecture description are components, connectors between them, and architecture configurations; these elements serve as categories to organize the framework. The framework represents a conceptualization of ADLs based on the literature thus far; as such, it provides an analytical tool to reason about ADLs, or, a *theory for analysis* [65]. The framework is constructed with an aim to be applicable to *any* ADL, hence, a high level of generalizability. Another example of a formal theory is a general theory of software engineering (GTSE) developed by Wohlin et al. [225]. The authors' theory is developed based on empirical observations of industry practice, and aims to explain how organizations can successfully develop software by balancing different types of intellectual capital. Project managers can use the theory to inform their decision making processes.

3.8 Computer Simulations

The eighth research strategy is *computer simulation*, also positioned in Quadrant IV. The goal of a computer simulation is to create a symbolic replica of a certain type of concrete system that can be executed by a computer [170, p. 87]. In a computer simulation of a real-world phenomenon or setting, everything is represented symbolically and created artificially. Where studies in natural settings (Quadrant I) can be costly or even impossible to conduct as they may involve a higher level of engagement from participants, computer simulations (which take place in a non-empirical setting) *"are like virtual laboratories where hypotheses about observed problems can be tested,"* before they are implemented in real-world systems [144]. Consequently, while simulations model a specific system or phenomenon, a simulation takes place in a non-empirical setting. That is, while variables can be modeled and manipulated based on the rules that are defined *within* the computer simulation, the researcher does not make any new *empirical* observations of behavior of outside actors in a real-world setting (whether these are human participants or systems); hence, computer simulations are positioned far away from point 'B' in Fig. 1. Tisseau used the term *in silico* to refer to "computerized calculations" [80, 208]. While prior empirical results can be used to create and calibrate computer simulations, this strategy does not lead to *empirical* results itself.

As a metaphor, we liken a computer simulation to a *forecasting system*, such as those used in weather prediction, which employ complex mathematical models of the atmosphere and oceans. Such systems are programmed to do a very specific thing based on a set of pre-programmed rules. The lack of further input by external actors

sets computer simulations apart from experimental simulations, which we compared to a flight simulator and a greenhouse. An example of a computer simulation is a study that simulates adaptive security in buildings [211]. Before actually implementing such a system in the real world (which would involve making many costly changes to the physical building), the authors performed a simulation as proof of concept. By analyzing a number of pre-programmed 'threat scenarios,' the researchers could evaluate the applicability and effectiveness of the approach as well as evaluate the classes of security requirements that could be handled by such a system.

4 APPLICABILITY OF THE ABC FRAMEWORK TO SOFTWARE ENGINEERING RESEARCH

To illustrate the use of different research strategies in software engineering research, we present examples from two different research areas: Global Software Engineering (GSE) and Requirements Engineering (RE). We selected these topics because they are well-established and important research areas within the software engineering literature, evidenced by their dedicated conferences (ICSGSE, RE). For both areas, we selected exemplar studies that clearly illustrate the eight research strategies. We emphasize that we do not intend to cast any judgment on the cited examples. However, for each example study we discuss some inherent limitations that are a consequence of the selected research strategy—rather than due to poor research design decisions of the authors. In addition to these two detailed examples, we also applied the framework to a sample of 75 articles in Springer's *Empirical Software Engineering* journal. Sec. 4.3 discusses details of the decision rules as well as a summary of the results.

4.1 Research Strategies in Global Software Engineering Research

GSE has been actively studied since the nineties, and research has focused primarily on the challenges associated with distributed development, whether as a result of offshoring or outsourcing strategies.

4.1.1 Field Study. A key issue in GSE is that of coordination of distributed teams as geographical, temporal, and socio-cultural distances give rise to a variety of challenges [2]. To investigate why coordination of distributed teams is so difficult and how associated challenges manifest in a real-world context, Herbsleb and Grinter conducted an in-depth case study at one division of an organization with teams in the UK and Germany [73]. The study's direct goal was neither to *evaluate* specific theoretical constructs affecting coordination (which were not well understood at the time of the study), nor to *improve* coordination in the organization. Instead, the authors aimed to develop a deep understanding of the intricacies of coordination of distributed teams in a real-world setting. Hence, the field study strategy was highly appropriate.

The results of this study discussed the means of coordination (e.g., component specifications and software processes) and their limitations, and barriers to informal communication (e.g., lack of unplanned contact such as conversations at the water cooler). The study concludes with a number of lessons learned; for example, Herbsleb and Grinter suggested bringing people from different locations together early on in a project. This type of insight would be unlikely to emerge from, say, a laboratory experiment.

This study captured specific events rooted in a concrete, real-world and thus realistic context. While high in realism, the study does not aim to generalize to a population of companies that conduct distributed development, although some of the recommendations may be useful to other organizations. Furthermore, the study did not *measure* any specific behavior nor did it establish any causal relationships. These are *inherent* limitations of field studies. Instead, this study's value lies in capturing a realistic setting, thus achieving its potential strength.

4.1.2 Field Experiment. Geographical distance has consequences for development activities of distributed teams. To ensure a high level of software quality, organizations can implement so-called "validation" activities, such as code inspections, peer review, and testing. In order to understand the impact of geographical distance on software quality when conducting such activities in a distributed fashion, Ebert et al. conducted a field experiment [49]. The authors investigated three factors that might impact the cost of rework: (1) the effect of co-location

on 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 projects. To evaluate these hypotheses, Ebert et al. used project data that the company had carefully collected over several years. To address the first two hypotheses, the data set was divided into different groups, based on specific parameters (e.g. whether or not inspections were done by co-located or distributed teams). For the third hypothesis, a number of changes were made to the development process, which were tracked in the project data. The authors found that inspections conducted by co-located teams significantly improved both efficiency and effectiveness of defect detection. Furthermore, the authors also found that providing coaching to teams (at a cost of 1-2% of the project budget) led to a reduction of rework cost. Finally, the various changes made to the development process in relation to teamwork and continuous build also reduced rework cost.

Ebert et al.'s study is an excellent example of a field experiment. Their study description addresses many potential threats to validity to ensure that we can have confidence in the findings. Nevertheless, the study has some inherent limitations. For example, one hypothesis was evaluated using a set of projects *"within one culture (i.e. Europe) and similar skill background that received a coaching effort of ca. 1..2% of total project budget"* [49, p. 302]. The precision of measurement of culture and background is limited—measuring latent constructs such as culture is a non-trivial task [76]. Furthermore, the natural setting of the projects is likely to have had many confounding factors that were not controlled for. The phrase "similar skill background" also suggests some latitude. Notwithstanding these inherent limitations, the study's strength lies in the rich set of data captured in a natural setting, achieving a very high degree of realism.

4.1.3 Experimental Simulation. Despite technological innovations that allow developers to interact using high-quality communication channels, there are still *"formidable barriers"* associated with distant collaborations due to factors such as a lower level of trust [19]. Bos et al. hypothesized that co-located individuals interact more amongst each other, forming an *"in-group"* (*H1*) [19]. Furthermore, they suggested that isolated individuals would form a separate in-group due to being excluded from the co-located in-group (*H2*). Finally, they argued that co-located individuals would outperform isolated members of the team (*H3*).

In order to evaluate these hypotheses, Bos and colleagues conducted an experimental simulation using an online multi-player game [19]. Rather than real-world development activities (i.e., software development tasks), the task at hand was to fill "orders" of colored shapes, e.g., Blue Square and Purple Circle. Each player could produce one type of shape, so in order to complete an order players were required to 'buy' and 'sell' shapes—hence, there was a need to build relationships, negotiate and collaborate. The use of shapes instead of real tasks allowed for an easy understanding of the simulation, but greatly reduced the realism of the task. In this simulation, some participants were co-located, whereas others were not—these were referred to as 'telecommuters.' The simulation was further constrained by limited resources and time.

The researchers found strong evidence for hypothesis *H1*: co-located participants had a strong tendency to collaborate with one another, rather than with those who were not co-located. Furthermore, the isolated telecommuters formed an in-group of their own (supporting *H2*). As for hypothesis *H3*, there was no significant difference in performance between co-located participants and the telecommuters.

This experimental simulation is clearly characterized by a contrived setting and unrealistic tasks. Furthermore, given that the study was conducted with a relatively small group of participants, the study's findings are not readily generalizable. However, these are inherent limitations of the experimental simulation strategy. Instead, the authors aimed to measure participants' behavior with a high degree of precision. In a realistic setting (i.e. a distributed software development organization), achieving this level of precision of participants' behavior would have been much more difficult, and also very expensive.

4.1.4 Laboratory Experiment. A system's software architecture plays a pivotally important role in coordinating distributed teams [72, 151]. Before a software system is implemented, its software architecture (SA) should be

evaluated as it has a major impact on quality attributes such as performance and reliability. Most SA evaluation methods (e.g., ATAM [93]) assume that the various stakeholders attending such reviews are co-located, which can be difficult or costly to achieve for globally distributed teams. Alternatively, SA evaluations can be conducted in a distributed setting supported by groupware tools. However, this raises the question as to whether the quality of distributed evaluations is as good as those conducted face to face. To evaluate this, Babar et al. conducted a controlled experiment [8]. Using 32 teams of three undergraduate students each, Babar and colleagues found that the quality of the scenario profiles developed by the distributed teams using groupware tool support were, in fact, of higher quality than those developed by co-located teams.

The setting of this study was clearly contrived: the authors set up an environment specifically for the purpose of this study. Student (instead of professional) participants were specifically recruited for this experiment, and the scope of the experiment was limited to only one activity of the architecture evaluation process (i.e., scenario development). The experimental tasks and instruments were also simplified, thus reducing realism of the context. Nevertheless, the study enabled the authors to achieve a high level of precision of measurement of the effectiveness of distributed architectural evaluations, and these findings could motivate field experiments to gauge whether such findings hold in a real-world setting.

Table 6. Examples of different research strategies used in global software engineering research

Strategy	Authors	Objective	Study setting	Study procedure	Findings
Field Study	Herbsleb & Grinter 1999 [73]	To understand why geographically distributed development is difficult to coordinate.	*Natural:* Two locations of a division of Lucent Technologies	18 interviews, archival sources, documents.	Coordination mechanisms; barriers to informal communication; lessons learned for multi-site development.
Field Experiment	Ebert et al. 2001 [49]	To investigate the impact of co-location, coaching, and teamwork and a continuous development process on software quality and cost.	*Natural:* Alcatel's Switching and Routing business unit. Data collected over a period of several years.	Comparison of inspections in co-located and distributed teams; projects within one culture and similar backgrounds with and without coaching; 'before' and 'after' introducing teamwork and continuous development process	Co-locating peer reviews improves defect detection; coaching within the project reduces cost of rework; teamwork and continuous build in the development process improves global project management.
Experimental Simulation	Bos et al. 2004 [19]	To study the effect of co-location, the presence of multiple sites within a large company, collaboration across multiple sites, and the influence of social networks in these collaborations.	*Contrived:* online multi-player game (the *Shape Factory simulation environment*). Participants recruited through a campus newspaper ad.	13 simulation sessions with 5 rounds each; 10 players per session, 130 participants in total.	Co-located participants collaborated more with each other than with telecommuters. The telecommuters also formed an in-group. No significant difference in performance between co-located individuals and telecommuters.
Laboratory Experiment	Babar et al. 2008 [8]	To study the impact of groupware support on the quality of software architecture evaluation deliverables.	*Contrived:* experimental tasks part of assessed course tasks. Participants received training on SA evaluation, tools.	Controlled experiment, AB/BA crossover design; 32 teams of 3 participants (3rd/4th year undergrad students of a SE course).	Quality of deliverables from the distributed meeting groups was significantly better than the quality of deliverables from the F2F meeting groups.

Continued on next page

Continued from previous page

Strategy	Authors	Objective	Study setting	Study procedure	Findings
Judgment Study	Iacovou & Nakatsu 2008 [78]	To investigate risk factors for offshore-outsourcing software development.	*Neutral:* systematically selected panel of experts at a variety of organizations based on their experience.	Delphi study, 15 experts, 3 rounds: identification; rating; rating feedback and revision. Interaction presumably online.	25 risk factors that could influence the success of an offshore-outsourced project, rated in importance by the experts; the top 10 are discussed in detail.
Sample Study	Ma et al. 2008 [125]	To investigate 3 issues in software development by Chinese software suppliers: language barriers, channels of communication, and working overtime.	*Neutral:* questionnaire sent out to companies by email.	Random sample of 2,000 from a database of approx. 6,000 Chinese software companies; 53 responses from 41 companies.	Language not a major obstacle; email used for development issues and face-to-face meetings to discuss management issues / requirements; reasons for overtime are requirement changes and underestimation of effort.
Formal Theory	Espinosa & Carmel 2003 [52]	To develop a conceptual foundation for future research on GSE.	*Non-empirical:* desk research without any direct empirical observations	Theorizing and conceptualization based on *Coordination Theory* and previous exploratory field research reported elsewhere.	A model of coordination costs due to time differences in dispersed software teams.
Computer Simulation	Setamanit et al. 2007 [160, 177]	To evaluate the choice of task allocation strategy and its impact on project duration.	*Non-empirical:* GSD simulator with which the researchers can model several factors, resulting in different GSE strategies.	For each of 3 strategies: 5 replications for each design point (7 factors, 2^7), resulting in 640 runs per strategy; 1,920 runs in total.	Increasing overlap of work hours contributes to shorter duration for module-based and phase-based strategy, and a longer project duration for FTS strategy.

4.1.5 Judgment Study. It should be clear by now that embarking on a GSE initiative is fraught with challenges. In the mid-2000s, numerous organizations had experienced such challenges, while outsourcing was still increasing in volume and importance [78]. This increased interest in outsourcing and offshoring can be explained by the fact that managers are continuously seeking ways to make software development faster and cheaper. However, at the same time managers may not be cognizant of the specific risks associated with GSE. Thus, Iacovou and Nakatsu set out *"to produce a set of project risks that specifically applies to offshore outsourcing"* [78, p. 90]. In particular, they adopted the judgment study strategy, implemented as a Delphi study *"to solicit and analyze the input of the expert panelists"* [78, p. 93]. The panel members were systematically selected: 57 project management professionals were invited to fill out a pre-study questionnaire to summarize their experience. After screening, 15 of them were invited to participate in the study. While the authors do not report on how they interacted with the experts, we assume this was done through postal mail or email (i.e. a neutral setting)—in any case, the authors focused on input from experts based on their experience, rather than specific software projects they were involved in. Through an iterative process of three rounds, this study identified 25 risk factors, which were ranked in importance according to the panel. The analysis indicated a statistically significant and high level of consensus regarding the risk factors.

This study offers excellent insights to other managers considering GSE initiatives. However, the study does not consider any specific context—the risk factors may or may not be applicable to a particular organization's context. This is an inherent limitation of the judgment study strategy. Furthermore, given the systematic selection of a relatively limited number of participants instead of a large, representative sample of participants, these findings

are not necessarily generalizable to the wider population of software development managers. On the other hand, this strategy did offer the researchers more control during the study through interaction in multiple rounds and interacting with the panel in a more intensive manner.

4.1.6 Sample Study. Whereas the judgment study discussed above investigated distributed development from an outsourcing *customer's* perspective, Ma et al. [125] took a complementary approach by investigating the *supplier's* perspective. In particular, as managers are adopting GSE initiatives to improve software development (either through reducing cost or time to market), Ma et al. investigated how well projects outsourced to Chinese suppliers actually performed in terms of language barriers, communication, and overtime work. To that end, they adopted the sample study strategy, using a questionnaire for data collection. Two thousand companies were invited, resulting in 53 responses eligible for analysis. Compared to judgment studies, the sample study strategy aims to achieve a higher level of generalizability of findings by gathering data from a large and representative sample. However, the number and complexity of questions that can be asked through a sample study is typically more limited than in judgment studies. When the number or complexity of questions becomes too high, the response rate may drop considerably. Overall, the authors found that language was not a major obstacle; email (asynchronous communication) was primarily used for development issues, whereas synchronous, face-to-face meetings were used to address management issues and product requirements. Key reasons for overtime work were changing requirements and suppliers' initial underestimation of the work.

This sample study achieved some level of generalizability, though in this case the specific sample was limited, considering that 2,000 companies were initially invited. However, this is always a challenge in conducting such sample studies. To indicate the representation of this sample: the respondent companies' size was distributed with 67% considered small (less than 50 employees), 26% medium (50-300), and the remaining 7% classified as large (over 300) and very large (over 1,000). Other limitations were more inherent to the selected strategy. The authors were, for example, not able to interact in-depth with the respondents to clarify responses or to ask follow-up questions (thus, precision of responses might have been low). In this study, the authors conducted a follow-up interview study with a number of respondents, but this is not part of the sample strategy and represents a separate research activity; also, this is only possible if respondents identify themselves. Another inherent limitation is that this study was not able to capture any specific context—thus, these findings must be interpreted with care as contextual factors may affect the performance of outsourced projects.

4.1.7 Formal Theory. Temporal distance is one of the distances in globally distributed teams, referring to the fact that teams work in different time zones. Time zone differences reduce the 'overlap' of working hours of distributed teams, and thus the potential to communicate through video link or teleconferencing. Asynchronous communication (e.g. email) is also affected as responses may be delayed several hours. This in turn introduces a number of challenges for project coordination. To investigate the implications of temporal distance for distributed teams, Espinosa and Carmel presented a theoretical discussion from different perspectives [52], leading to a model that affords a unified view of coordination challenges in time-separated contexts, which provides a conceptual foundation. As the authors suggested, this foundation provides the basis for simulation research, experimental research and field research. Indeed, at the time of writing, this article has been cited over 160 times, suggesting that other researchers have used this foundation for their studies.

Formal theory development studies such as these are important as they advance the field by offering a new conceptual lens to design future empirical studies. Such studies lack contextual realism, however, as they do not rely on any observations or data gathered in real organizations. Furthermore, given that such studies are conducted in a non-empirical setting, there is no observed behavior, either from participants or from software systems. On the other hand, the newly proposed conceptual framework abstracts reality, aiming at a high degree of generalizability.

4.1.8 Computer Simulation. Beside drawbacks, temporal distance also offers benefits, such as *Follow-the-Sun* (FTS) development. As one team finishes the workday, the next team can continue to work on the same task. Some project managers estimated that by adopting FTS, the time-to-market can be reduced by 20% to 35% [24, p. 33]. Besides FTS, there are two other strategies to configure task allocation in a distributed development project: *phase-based* and *module-based* development [24, p. 130]. Following the module-based strategy, each development site develops a software module from start to finish. In a phase-based configuration, a development site performs a particular phase of the software development life cycle. For example, site A may perform design, followed by implementation at site B. This raises the question: *which task allocation strategy is most optimal?*

To investigate, Setamanit et al. conducted a computer simulation to evaluate these task allocation strategies [176, 177]. The researchers first investigated the 'ideal' circumstances, without any communication and cultural barriers. This was modeled by excluding all factors affecting productivity and defect injection rate from the simulation. In this case, using a FTS strategy, the development cycle time was 70% shorter than single-site development. The module-based approach took slightly longer than the FTS strategy, and the phase-based strategy was very similar to a single-site development scenario. However, the results differed significantly when taking into account the various GSE-specific factors. In this case, the FTS strategy took 37% *longer* than a single-site development scenario, while the module-based strategy resulted in the shortest development cycle time. The development cycle time resulting from the phase-based strategy was similar to that following a FTS strategy.

Using the computer simulation strategy, the authors were able to evaluate a number of different scenarios. Evaluating these scenarios in a real-world setting would most likely have proven infeasible or too costly. Instead, the researchers could perform this work at their desk. However, one inherent limitation of this approach is that no real-world behavior can be observed, given that a computer simulation is conducted in a non-empirical setting. The simulation is simply a model, but may not accurately represent reality.

4.2 Research Strategies in Requirements Engineering Research

In this section we illustrate the different research strategies with a second research area: Requirements Engineering (RE). RE has long been one of the core areas within software engineering research, with a first special issue in *IEEE Transactions on Software Engineering* in 1977 [167].

4.2.1 Field Study. RE has been cited as a critical activity for any software project. The rise of GSE initiatives has led to new challenges for RE, some of which were highlighted in Sec. 4.1. To investigate these new challenges, Damian and Zowghi conducted a field study at one large multi-site organization [37]. One of the authors spent several months on-site at the case organization to develop an in-depth understanding of the specific context. Data were collected through document study, observation, and interviews. The authors report on specifics of the context, including the organizational structure and collaboration technologies. The study reports four key problems in distributed development, and eight specific challenges for RE. These challenges are described in detail and illustrated with examples from the specific context of the case study organization—thus achieving a high degree of realism. The authors conclude that improving communication would significantly reduce the impact of global collaboration on requirements management.

Being on-site at an organization for an extensive period of time greatly helps to achieve a high degree of realism and to capture many details of the organizational context. This is simply not possible when conducting, for example, a sample study of organizations. On the other hand, the rich contextual findings are not necessarily generalizable to other contexts, although some of the challenges identified were also confirmed in other studies. Furthermore, the authors gathered data through observations and interviews, but the lack of precision of measurement of the multiple variables present in a natural setting prevent any inference of causal relationships.

4.2.2 Field Experiment. Requirement defects are problems with a software product that make it unfit for its surroundings or for its users. Preventing such defects can greatly benefit the quality and satisfaction of end-users. Lauesen and Vinter conducted a study to *"find cost-effective ways to avoid requirement defects in the products"* [111]. The authors adopted the field experiment strategy, by means of Action Research [111, p. 38]. The authors first studied one project to establish a baseline in order to understand how requirement defects occurred. To improve the RE process, Lauesen and Vinter studied 44 techniques from which ten were deemed worthy of further consideration. Due to various constraints outside the researchers' control, only one project adopted two of the ten techniques. After the project was finished, the researchers studied the defect reports and the final product, but no improvements could be observed due to the fact that the contexts of the projects were too different, although the authors did observe some other effects that suggested improvements.

Interestingly, the authors reported that: *"we hadn't imagined how difficult it was to compare two very different projects"* [111]. This is a significant characteristic of field experiments; the researchers simply could not precisely measure differences due to the natural research setting, and faced constraints outside their control. Another inherent limitation that became clear was the generalizability of findings. Each software project is different, which makes comparison problematic, even within the same organization.

4.2.3 Experimental Simulation. Identifying the requirements for a new software system is very important in order to ensure that the system will be accepted by its future users. However, this has proven very difficult, which also explains the prominence of the RE field in the software engineering domain. Before designing new systems that will operate in complex environments to help users in their decision-making process, it is important to establish how the system will be used. Unfortunately, in highly dynamic and complex environments it is not possible to simply ask users to describe their decision-making process [116]. Therefore, Lerch et al. set out *"to discover* how *and* why *end-users make decisions"* [116, p. 346] (emphasis in original) through an experimental simulation. The specific goal was to assess computer support needs in the US Postal Service. As the authors asserted, an experimental simulation facilitates the assessment of system features without building these systems and without the need to interfere with production environments.

The authors implemented a simulation which used real-world data that was collected prior to the experimental simulation. During the simulation, several types of data were recorded in order to compare the behavior of novices and experts. Data were recorded through keystroke loggers and verbal protocols. The simulation tool offered a considerable level of precision to measure behavior, as Lerch et al. describe [116, p. 350]:

> *"we were able to observe the behavior of supervisors with different levels of experience under identical environmental conditions. [...] In addition, by controlling the environmental conditions, it was possible to collect data (...) to compare the performance of experts and novices."*

This experimental simulation took place in a contrived setting facilitating a relatively high degree of precision to measure a number of variables that capture the different types of behavior of participants. Designing such a study requires, however, considerable effort: in this case the authors went to great lengths to implement a simulation that *"mimicked the real world"* [116, p. 350]. Prior to implementation, the authors had spent considerable time interviewing and observing staff at the mail facilities.

4.2.4 Laboratory Experiment. Once requirement specifications are written, different approaches can be used to validate these. One of the goals in validating requirement specifications is to detect any faults, which is important for the quality of the resulting software and user satisfaction (as discussed in Sec. 4.2.2). Porter et al. conducted a multi-trial experiment to compare alternative fault detection methods [159]. The experiment was conducted twice, in both cases with 24 graduate students who took a formal methods course.

One of the study's key findings is that scenario-based methods led to a higher fault detection rate than ad hoc and checklist methods. While the study achieved a high degree of precision of measuring the various variables,

the study lacked generalizability and realism—but, these are two inherent limitations of the laboratory experiment strategy. The authors declared several threats to validity: the subjects (i.e. students) were not representative of software professionals due to their limited experience and motivation to participate in the study; the specification documents may not have been representative of professional documents; and, the inspection process may not have been representative of industry practice.

4.2.5 Judgment Study. Many organizations adopt off-the-shelf Enterprise Resource Planning (ERP) packages, which have to be tailored not only to their own organizational setting but also to that of their customers. Such inter-organizational ERP projects have their own set of requirements, and identifying and implementing them can be a challenging endeavor [39]. Daneva proposed a set of 12 RE practices to support such operations [38]. To evaluate them, Daneva and Ahituv [39] adopted the judgment study strategy, conducting a focus group of 10 systematically selected practicing ERP consultants. This study validated all 12 practices.

This judgment study depended on the expertise of ERP consultants who were presented a very specific "stimulus," namely the 12 proposed RE practices for inter-organizational ERP projects. Conducting such an evaluation with a sample study would have been an alternative strategy (aiming at a higher degree of generalizability through a larger, representative sample of experts), but this would have reduced the degree of interaction needed to explain and qualify the 12 practices, potentially leading to misinterpretations. This clarifies the benefit of a judgment study over a sample study in this case. Interestingly, Daneva and Ahituv acknowledged the lack of a realistic context and suggest adopting a field study strategy as their next step: *"Our immediate step will be to run a case study in two organizations to develop an understanding of the context"* [39].

4.2.6 Sample Study. RE is an active research area, and it is important to regularly collect data about the state of practice so that researchers understand the challenges that professionals face, and what solutions could benefit them. Neill and Laplante identified a lack of contemporary data about actual requirements elicitation in practice [146]. Without such data, researchers may wrongly assume how software professionals operate. Therefore, Neill and Laplante adopted the sample study strategy by conducting an industry survey to assess the state of practice of RE [146]. Data were collected through an online questionnaire with 22 questions. From the 1,519 invited persons, 194 completed the survey. Respondents worked in a wide variety of different companies including multinational companies, operating in various domains such as telecommunications, aerospace, and manufacturing. The range of positions in their respective organizations was as diverse, from system designer to executive. The findings indicated that over 50% of respondents used scenarios or use-cases in the requirements elicitation phase, whereas 30% indicated to be using object-oriented analysis [146].

This sample study provided data on RE practices from a wide range of software organizations, and thus this study scores high on generalizability. However, no variables were manipulated to investigate any effects, nor was the survey able to capture any specific context. Thus, while we learn little about which RE practices are appropriate for ERP projects (cf. Daneva and Ahituv's judgment study [39]), or the various challenges that distributed development introduce in a specific organization (cf. Damian and Zowghi's field study), we learn about some trends manifested in a wide array of organizations.

4.2.7 Formal Theory. In recent years, researchers have also addressed the role of creativity in RE. To better understand this role, Nguyen and Shanks developed a theoretical framework [147]. The framework development draws on the wider creativity research literature and the literature on creativity in the RE field. The resulting framework consists of five elements: product, process, domain, people, and context. Using Gregor's taxonomy [65], the authors labeled their framework a *"theory for analyzing"* [147, p. 661], and in this case, to analyze RE creativity research. The authors identified two main implications for further research and practice. First, they called for empirical studies on how to integrate the framework elements into RE methods to support creativity. Second, they recommended that organizations establish an environment to encourage creative people.

Table 7. Examples of different research strategies used in requirements engineering research

Strategy	Authors	Objective	Study setting	Study procedure	Findings
Field Study	Damian & Zowghi [37]	To investigate the impact of distributed stakeholders on RE activities in GSD.	*Natural:* first author spent 7 months on-site at an organization.	Document study, observation, interviews	Four major problems and 8 specific challenges related to requirement engineering activities in distributed development.
Field Experiment	Lauesen & Vinter [111]	To identify a cost-effective way to avoid requirement defects.	*Natural:* company staff and researcher collaborated on-site, using real products to evaluate new approaches.	Action research, data include defect reports, time spent, usability issues, timeliness of the project, product sales.	Several conclusions, incl.: scenarios and early usability testing are beneficial techniques; classifying defects according to the source of the defect was not helpful.
Experimental Simulation	Lerch et al. [116]	To identify the computer support needs of automation staff in a large organization.	*Contrived:* Simulation environment with experimental stimuli that were previously collected.	3 distinct groups of 3 people each to gauge how level of expertise and circumstances affects behavior. Participants received training.	Insights into different information needs and search strategies and decision making strategies depending on users' expertise. Insights on performance with feedback / feedforward.
Laboratory Experiment	Porter et al. [159]	To investigate the hypothesis that scenario-based inspections are more effective than ad hoc inspections.	*Contrived:* classroom laboratory exercise with graduate students doing a course in formal methods.	Measurement of effect of detection methods (ad hoc, checklist, scenario) on 4 dependent variables incl. fault detection rate.	4 key findings, incl.: scenario-based method leads to higher fault detection rate than other methods; scenario reviewers were more effective at detecting faults the scenarios were designed to uncover.
Judgment Study	Daneva & Ahituv [39]	To evaluate a set of 12 practices based on feedback by ERP practitioners.	*Neutral:* dedicated meeting room with seating around a table.	10 consultants from 7 ERP services firms, selected based on their interest and expertise.	All 12 practices were observed by several of the panel experts.
Sample Study	Neill & Laplante [146]	To investigate the state of practice of requirements engineering in industry.	*Neutral:* web-based questionnaire.	22 questions; participants drawn from a Penn State School of Graduate Professional Studies database; 194 responses from a population of 1,519.	Findings include organization and participant characteristics (various domains; participants held variety of positions); software development life cycle model (agile, waterfall, etc.); RE techniques.
Formal Theory	Nguyen & Shanks [147]	To develop an understanding of the role of creativity in RE.	*Non-empirical:* no empirical observations, but derivation of a conceptual framework from literature.	General creativity literature, requirements engineering creativity literature.	A theoretical framework that offers RE researchers a basis to incorporate creativity in RE methods and techniques.
Computer Simulation	Höst et al. [77]	To investigate bottlenecks and overload in RE processes.	*Non-empirical:* a discrete event simulator was implemented in SDL.	Four simulation scenarios with different parameter values to model different circumstances.	Two ways were identified to avoid congestion: (1) increase number of staff or productivity, (2) decrease the rate of new requirements e.g. through prioritization.

This study did not result in any empirical observations, but through a careful derivation of a framework based on extant research on creativity, the authors developed a theory for analyzing the role of creativity in RE—this in

turn can be used as a foundation for future empirical studies to investigate how each of the framework's five elements manifests, for example. Published in 2009, this study has been cited well over 100 times, suggesting a considerable number of authors have extended this strand of research.

4.2.8 Computer Simulation. Much research in the RE field has focused on eliciting requirements for *bespoke* software projects, with far less attention paid to so-called *market-driven* software development [77], whereby commerical off-the-shelf (COTS) software packages are regularly released—for example ERP packages mentioned earlier. Software organizations that make such COTS components are also looking to improve their development processes, but there are many ways to optimize such processes to overcome bottlenecks.

Höst et al. [77] point out that evaluating process improvements could be done through field experiments, for example, but that this would require considerable effort. Instead, Höst et al. adopted the computer simulation strategy to investigate situations that lead to bottlenecks. Using a series of simulations, the authors were able to evaluate how different changes to the processes led to an improved process (i.e., removal of bottlenecks). In particular, the computer simulation was implemented as a discrete event simulation model. The authors evaluated four scenarios, each with a different configuration of parameters. The authors concluded that "congestion" in the process can be avoided, either by increasing the number of staff or productivity, or, by limiting the rate of new requirements through, for example, requirements prioritization. However, it is important to note that while such computer simulations may suggest how changes to the process might work *in theory*, empirical studies are needed to actually evaluate such interventions in practice, for example through field experiments. This lack of a realistic setting is an inherent limitation of computer simulations.

4.3 Analysis of a Sample of Studies

Our analysis above is based on a convenience sample; we selected studies that exemplify the eight research strategies. In order to demonstrate a wider applicability of the framework, we analyzed all articles published in 2017 in Springer's *Empirical Software Engineering* journal. (Table 10 in Appendix A presents the results.) Our analysis shows that all studies can be mapped to the ABC framework. Table 8 presents the count of each research strategy; several articles present multiple studies, which is why the total count ($n = 85$) is higher than the number of articles ($n = 75$). The table shows that Sample Studies ($n = 38$) and Laboratory Experiments ($n = 30$) are most widely used. Field Studies were reported in 10 articles, and only a few articles reported Experimental Simulations and Judgment Studies. We found no instances of Field Experiments, Formal Theories, or Computer Simulations in this particular sample.

Table 8. Distribution of research strategies of the analyzed sample

Strategy	Count
Field Study	10
Field Experiment	0
Experimental Simulation	4
Laboratory Experiment	30
Judgment Study	3
Sample Study	38
Formal Theory	0
Computer Simulation	0

The ABC framework in Fig. 1 organizes the eight archetypal research strategies along two dimensions: obtrusiveness and generalizability. Below we offer a set of questions to position a study in a specific area of the circumplex.

To map a study to the circumplex, the first two questions to ask are:

- Does this study focus on a *particular instance* of a phenomenon, or does it aim to *generalize*?
- What is the *level of obtrusiveness* of the researcher in the research setting?

Furthermore, the eight strategies are placed in four 'quadrants' with boundaries at a 45 degree angle (not bounded by the two dimensions mentioned above) that characterize the research setting. That is, has the research been conducted in a:

- *naturally occurring setting* that existed before the researcher entered (Quadrant I);
- *contrived setting* that the researcher set up specifically for the study (Quadrant II);
- *neutral setting*: one that is not related to the research findings, or one that the researcher has actively set up to be neutral, so as not to affect the research findings (Quadrant III);
- *non-empirical or theoretical setting*; that is, the research has been conducted without making any empirical observations (Quadrant IV).

The research strategy adopted can be identified without too much effort in most studies. However, in some cases it requires careful reading. We give three examples below.

- Rojas et al. [165] sought to evaluate the effectiveness of whole test suite generation. They conducted an empirical study on 100 Java classes, and compared three different coverage criteria (line coverage, branch coverage, and weak mutation testing). This is an example of a Laboratory Experiment, because the researchers were clearly interested in a precise comparison of three different approaches. The research setting was specifically created for the study's purpose (hence, a contrived setting). While the researchers used a 'sample' of 100 Java classes (which some might associate with a Sample Study), the goal was not to seek maximum generalizability to a population, but rather to precisely measure the behavior of three different approaches as measured in effectiveness.
- Vitharana [215] investigated the impact of defect propagation at the project level during early lifecycle phases. The article sets up a set of hypotheses which are tested using a large data set consisting of inspections conducted over a period of four and a half years at one company. To evaluate the hypotheses, a data set from one company is used; while this might suggest it is a study of a particular organization (i.e. a Field Study), the goal of this study is to establish general relationships—hence, we classify this as a Sample Study. The fact that the data set originated in one company is of secondary importance. In practice, using data from a range of organizations is very challenging because companies may not collect the same type of information.
- Spinellis [194] presents an analysis of the history of Unix. The analysis is based on the complete development history of the operating system, which might suggest a Sample Study. However, the study focuses on one particular software program; rather than seeking generalizable findings, this study aims to understand one specific context of software development, namely the Unix operating system. We classify this as a Field Study, despite the fact that the research was conducted as desk research, not "in the field." One obvious reason for this is that this is an archival study, because development took place several decades ago.

5 DISCUSSION AND CONCLUSION

5.1 Metaphors and Research Settings in Software Engineering

For each research strategy, we presented a metaphor representing that strategy within its research setting (see Sec. 3 and Table 5), some of which were proposed by others [135, 170]. Metaphors are widely used in the software engineering literature [181] and practice [106]; indeed, the term "software engineering" itself can be considered a metaphor [21]. Metaphors are powerful pedagogical devices that can help to convey the essence of a term or entity. However, they are usually limited in the extent to which they can fully capture the similarity with the

real-world entity they represent. Here, we briefly discuss these metaphors and the key benefits and drawbacks of each research strategy (see Table 9).

Quadrant I Natural Settings: *Field studies* take place in natural settings, which we liken to a *jungle*, as it allows a researcher to investigate a real and concrete instance of a phenomenon. Findings are grounded in realistic or concrete settings that exist independent of the study. However, in order to retain the undisturbed setting, the researcher cannot manipulate any variables or properties, as this would disturb the naturalness of the setting. *Field experiments* are very similar in that they take place in a natural setting, but a researcher manipulates some variables or to administer different treatments to different projects or participants. However, this manipulation reduces the realism of context somewhat. We propose the *nature reserve* metaphor—like a jungle, it represents a natural setting, yet it allows a researcher to intrude on the setting by making changes to the "living circumstances" of some inhabitants. In software engineering projects, researchers are limited in natural settings by the willingness of participants or projects and their managers to participate in a study that intrudes on their work environment. It can also be very costly and complex to set up field experiments.

Quadrant II Contrived Settings: In the *experimental simulation* strategy, the researcher's intrusion in the research setting increases a bit more, and we borrow the *greenhouse* metaphor [170]. In a greenhouse, a researcher has a controlled environment that simulates a particular type of concrete setting, yet the researcher aims to elicit a somewhat realistic flow of events (e.g. the growth of a plant) rather than discrete trials (e.g. a reaction of two chemicals) more typical of laboratory experiments. Actors may behave in natural ways, although they will be aware of the contrived setting. Using an alternative metaphor of a *flight simulator*: there are no real consequences to crashing a plane in a simulator. The *laboratory experiment* strategy is characterized by an even higher level of intrusion by the researcher; rather than observing "natural" behavior in a contrived setting as found in an experimental simulation, in a laboratory experiment there is a very careful measurement of variables through a number of "runs" or trials, whereby the sequence of events is fully controlled. We borrow the *test tube* metaphor (ibid.).

Quadrant III Neutral Settings: Both *judgment studies* and *sample studies* are conducted in neutral settings. In a judgment study, the researcher is interested in capturing responses from a panel of participants based on some potentially complex stimulus. To facilitate this, the researcher may actively "neutralize" the setting, very similar to a *courtroom* so that judges and jury members are not distracted by the environment. In a sample study, on the other hand, the setting also bears no effect on the responses, but interactions between the researcher and the actors (either humans, systems, or other design artifacts) are much simpler. Hence, we liken the sample survey to a *referendum*.

Quadrant IV Non-Empirical Settings: Finally, the *formal theory* and *computer simulation* strategies are non-empirical strategies. Neither offers an opportunity to observe real-world behavior or interactions. We compare the development of formal theory to solving a *jigsaw puzzle* in that the researcher carefully constructs a model that helps to predict or explain a phenomenon, based on a thorough understanding of existing literature, but also through *"creative insight"* [170]. A computer simulation, on the other hand, is a symbolic representation of a concrete class of system that can be configured in a range of ways, allowing researchers to "run" a series of scenarios. In that sense, we argue that a computer simulation is very similar to *forecasting systems*, such as used for weather prediction and stock markets.

5.2 The A, B, and C of Software Engineering Research

The ABC framework positions eight archetypal research strategies that SE researchers can use, in particular for what we have termed knowledge-seeking studies (see Sec. 1). The framework is based on earlier work in the social sciences [131, 133, 134, 170, 217] which we have adapted and operationalized for a software engineering

Table 9. Software engineering research strategies, metaphors, and evaluation considerations

Strategy	Metaphor	Essence	Evaluation considerations
Field Study	Jungle	Facilitates the study of real-world actors (people, systems) and their behaviors in a natural setting that is not manipulated by the researcher. High potential to capture realistic settings and a high degree of detail of a particular system and context.	Not suitable to investigate statistical relationships, or to otherwise manipulate variables, nor for findings that hold for larger populations.
Field Experiment	Nature reserve	Facilitates the study of effects of a modification of properties of a studied entity or phenomenon that occurs in a natural setting, i.e. pre-exist independent of the researcher. Potentially very costly to set up due to complexity of natural settings.	Limited level of precision of measurement; results not generalizable, but strongly linked to the specific setting due to confounding variables that are very difficult to isolate.
Experimental Simulation	Greenhouse, Flight simulator	A contrived setting that simulates a specific class of real-world systems that to some extent resembles reality. Temporal flow of events depends on the simulation environment and actors' behavior, which allows for observing more natural behavior than a laboratory experiment.	Reduced level of realism compared to field experiments due to the contrived setting: behavior of actors may reflect that in natural settings, but consequences for actors lack realism which may affect their behavior.
Laboratory Experiment	Cleanroom, Test tube	A controlled setting where behavior of actors (humans or systems) is carefully measured through a number of discrete trials to establish effects or conduct comparative analyses. Maximum potential to capture precise measurement of variables (high internal validity) due to potential to isolate of confounding factors.	Studied relationships and variables are more abstract due to the contrived and 'sterile' nature of the research setting. The setting is more artificial than for experimental simulations.
Judgment Study	Courtroom	Facilitates study of responses or behavior of actors that bears no relation to the research setting, which is neutral or actively "neutralized." Allows for more complex questions and interactions between researcher and respondents.	No concrete or natural setting which prohibits capturing direct observations of phenomena.
Sample Study	Referendum	Facilitates data collection from a representative sample of a population (human or non-human, such as systems or design artifacts). Maximum potential to generalize findings to a wider population; rather unobtrusive research strategy. Researcher must work with whatever data is collected.	Limited precision of measurement: questions asked of, or about, the sample tend to be 'simple'; limited opportunity for 'complex' interaction between the researcher and subjects. Research setting offers no realistic context.
Formal Theory	Jigsaw puzzle	The careful and justified construction of a theoretical model that represents one view of a phenomenon, which helps to analyze or explain the real world. Model generic behavior for a range of classes of populations (humans or non-human artifacts), which serves to make predictions or explanations about the real world.	Theoretical models do not generate new empirical observations, though may inform future empirical studies.
Computer Simulation	Forecasting system	Represents a symbolic replica of a concrete real-world system where all configurations and variables are pre-programmed. Useful to run a large number of complex scenarios to explore a solution space, which might not be feasible to do manually.	All simulation rules are pre-programmed: no new *empirical* (i.e., real world, as opposed to simulated) behavior is observed. Due to concrete implementation, limited generalizability.

context. The ABC framework introduces a new approach to consider SE research studies which can be used in at least six ways:

- **Research Tutorial.** The ABC framework offers a holistic overview to novice researchers who may be overwhelmed by the numerous research methods and techniques on offer. While several books and articles in the SE literature provide overviews (see Table 4), these do not provide a holistic framework that can position the various methods available to the SE research community. Based on two dimensions that are widely considered to be key in choosing a research design, the framework can help understand the purpose, potential strengths and essential weaknesses of the eight archetypal research strategies.
- **Research Program Design**. The ABC framework can also be used, by both novice and experienced researchers, to design research programs. Doctoral students who conduct a series of studies for their dissertation might wish to select a different research strategy for each. For example, studies could adopt a strategy from different quadrants, thereby designing studies to take place in different research settings. By so doing, the researcher can investigate a topic through triangulation of strategies, thereby overcoming inherent limitations inherent in one strategy by adopting an alternative strategy. In this way, for example, more complete research coverage of an existing research topic could be achieved by selecting research strategies that had not been previously applied. Appendix B presents two example scenarios to demonstrate how novice researchers could design a program of studies that use different research strategies.
- **Literature Classification.** The ABC framework draws clear distinctions between different research strategies, each with clearly defined limitations. Given the different types of evidence that each research strategy can produce, the framework offers, in our opinion, a good structure to classify extant literature in, for example, systematic literature reviews. Also, because the ABC framework is grounded in two key dimensions that are important in research design, the framework offers a reference point for terminology that may help researchers to describe their studies. In Sec. 2 we gave two examples [17, 192] where the authors' initial characterization (and terminology) of their study led to misunderstandings and misinterpretations.
- **Research Reporting and Evaluation.** Besides merely classifying extant literature, the strengths and weaknesses of published studies can also be better evaluated and assessed, not only in the peer review process but also after studies are published. Even today, there is a continued concern with, for example, the lack of generalizability of exploratory case studies (a typical method in the field study strategy, and very common in SE research). The ABC framework clearly defines the maximum *potential* strengths of studies (indicated by the markers A, B, and C in Fig. 1) as well as some inherent limitations (see Table 5), which may provide guidance to reviewers.
- **Research Diversity.** The eight research strategies defined in the ABC framework each have their unique strengths and limitations, yet not all research strategies are widely known or used. While diversity of research methods used in the SE field has increased since Glass's observation of narrowness of research approach in SE, mentioned in the introduction [63], we believe an increased cognizance of how the various research strategies differ will encourage researchers to more consciously adopt diverse strategies. Additionally, we note that the framework also positions two non-empirical research strategies—formal theory and computer simulation—each of which offer specific strengths, and offer opportunities for SE researchers to study phenomena from a different perspective. Again, this contributes to potentially more thorough investigation of research topics.
- **Study Integration and Synthesis.** Several authors have lamented the lack of integration of individual primary studies in software engineering (see, for example, Cruzes and Dybå [31]). While the primary studies can result in useful findings, there is a need to study topics from more than one angle. On the basis of our two worked examples (see Sec. 4), which identified exemplar studies for each research strategy in two important SE areas, we have demonstrated how closely related research questions can be addressed with diverse strategies. Together, these findings can converge in such a way that we can develop theories within the SE domain.

The eight research strategies each have their limitations as discussed throughout this paper and illustrated in the two examples discussed in Sec. 4. These limitations are *inherent*, in that they cannot be ameliorated by any tactics that a researcher might deploy. For example, the fact that the research setting in a laboratory experiment is highly contrived and unrealistic is an inherent limitation. It cannot be "fixed," and nor should a researcher apologize for this fact. Likewise, a sample study should not be criticized on grounds of lack of realistic context—it is an inherent limitation of sample studies.

An important consequence of the observations with respect to the strengths and weaknesses of different research strategies is that the SE research community needs better guidelines to more fairly evaluate the quality of research studies, which should be evaluated according to how well their *potential* strengths have been achieved and the degree to which incidental or operational limitations have been identified and ameliorated. An incidental or operational limitation is one that might have been prevented by a researcher. For example, a sample study with a very small number of respondents (say, 10, which under any circumstances is a very small number) could be evaluated less favorably given that its potential strength, achieving a high degree of generalizability to a large population, has not been achieved. When evaluating a field study, reviewers should not lament the apparent "lack of control or generalizability" in the study, because those are its inherent weaknesses. Rather, such studies with a high degree of realism should be assessed on the extent to which potential strengths have been realized, and the extent to which incidental limitations have been addressed. Likewise, it would be unfair to criticize a laboratory experiment because of its lack of realism. Rather it should be assessed according to the extent that it has realized its *potential* strengths, such as the degree of control of variables (construct validity, internal validity). This is not to say that calls for more realism, such as by Sjøberg et al. [188] can be ignored, but rather both researchers and referees should acknowledge the inherent limitations of the adopted research strategy.

5.3 Limitations of the ABC Framework

Given the socio-technical nature of the software engineering field we argue that the framework, though rooted in the social sciences, is also useful to SE research. We have demonstrated the fit of the ABC framework on two worked examples in Sec. 4. In addition, we analyzed a sample of 75 articles published in 2017 in *Empirical Software Engineering* (see Appendix A); we classified all knowledge-seeking studies using the ABC framework. Notwithstanding the fit of the ABC framework, some limitations of the framework should be borne in mind.

First, the ABC framework does not provide guidance for specific methods—Table 4 provides an overview of selected resources for specific methods, and so this article complements this existing guidance. Instead, the ABC framework provides a holistic overview of different research strategies, each with specific characteristics, positioned along two key dimensions (see Sec. 2.3) that facilitates a systematic comparison. As a researcher selects a specific method (such as a controlled experiment, or ethnography, for example), the researcher still needs to consider a variety of research design considerations that are specific to those methods. This is true also during operationalization of a study: for example, a sample study can involve human respondents or using a sample of software process artifacts (typically gathered through repository mining). A researcher must consider the specific challenges associated with each of these data collection methods (i.e. challenges associated with repository mining [92] or with conducting sample studies with human respondents [97]).

Second, the ABC framework presents what we call *archetypal* research strategies, which is to say that other researchers may design studies that do not fall cleanly within one of the octants of the framework. There are certain research methods that represent a compromise between different strategies. For example, Jansen discusses the "qualitative survey" and contrasts it with what he refers to as the "statistical survey" [82]. The latter is equivalent to what we refer to as *sample study*. Jansen defines the goal of a qualitative survey to study diversity of a topic within a population (ibid.). While the term "survey" might suggest the qualitative survey falls within the sample study strategy, the ABC framework makes clear that its position may vary depending

on the goal. A qualitative survey may be classified as a judgment study, in particular when a researcher wishes to explore diversity of opinions or responses based on a given stimulus (akin to a Policy Delphi study [105]). Alternatively, as Jansen points out, a qualitative survey could be part of a Grounded Theory study [82]. Rather than mapping methods to the various archetypal research strategies, we believe it is more fruitful to represent the ABC framework as a device that represents *polarity*, within which methods can vary among the three distinct goals of generalization, precision of measurement of behavior, and of capturing a realistic context. As such, it helps a researcher to carefully consider the trade-offs that are represented by the framework.

Third, the ABC framework is an *alternative* view to software engineering research, not necessarily the best view. Previous classifications such as by Shaw [182] and Wieringa et al. [221] may be a better fit for positioning studies within a given research program, in particular for solution-seeking studies. As we pointed out, establishing a general taxonomy is very challenging. Rather than presenting the ABC framework as a final solution to classify research methods, we prefer to position it as a device to reason about, and to design research studies. Researchers can consider the goal of their study, which can be to seek generalizability over a population of actors (developers, applications, defects, etc.), to investigate relationships and behaviors through precise measurement and manipulation of variables, or to capture a high degree of realism by studying specific contexts (e.g. a specific organization or system).

Finally, as outlined in Sec. 1, the ABC framework applies to primary studies only, and specifically those that we have labeled *knowledge-seeking*. While we suggest that primary studies can identified as knowledge-seeking or solution-seeking studies, there are numerous articles that cannot be classified as either, in particular methodological papers (e.g. [92, 161]), which includes this article. Therefore, the ABC framework is not useful to for analyzing or understanding such methodological papers.

5.4 Conclusion

The software engineering research community has made considerable progress in terms of the quality of studies that seek knowledge and understanding. Over the last several decades, the community has reflected on the way it conducts research, and the methods to do that research. There are numerous guidelines for employing specific methods, and the variety of research methods has increased without doubt. Nevertheless, there is still some confusion about terminology, and the various overviews of research methods are a"mixed bag" in that the various methods identified have not been carefully positioned in relation to one another. A holistic view of the landscape of research strategies to generate new knowledge and understanding has been missing so far in the SE research community. In this article, we adopt a framework from the social sciences which we have labeled the ABC framework—as such it contributes to the literature on research methodology for software engineering. The ABC framework represents a holistic overview of eight archetypal research strategies which is oriented around two key dimensions: the level of obtrusiveness of a study, and the generalizability of the findings. The framework also clarifies the inherent weaknesses and potential strengths of the research strategies, which facilitates a better understanding of the trade-offs between threats to internal and external validity, for example.

In this article we have tailored and operationalized the ABC framework for a software engineering context, and illustrated how the eight archetypal research strategies manifest in two key software engineering areas (GSE, RE). In addition, we have analyzed a sample of all articles published in 2017 in *Empirical Software Engineering: An International Journal*, which demonstrates that the framework is generally suitable to reason about research strategies of knowledge-seeking studies in SE. Finally, we discussed a number of ways in which future SE research can benefit from the ABC framework.

ACKNOWLEDGMENTS

We thank the three anonymous reviewers for their detailed feedback, which has led to a better article.

A SAMPLE ANALYSIS OF RESEARCH STRATEGIES

In order to demonstrate a general applicability of the framework, we conducted an analysis of all articles published in 2017 in Springer's *Empirical Software Engineering: An International Journal*. We selected this journal because it emphasizes empirical research, which is more likely to be "knowledge-seeking" than solution-seeking. The analysis included 75 articles; we excluded secondary studies (mapping studies, systematic reviews, and case surveys), editorials, methodological papers, and *errata*.

Table 10 presents the detailed results of our analysis. For each study, we list the authors, title, and predominant research strategy, followed by a brief description of the study. Several articles present solution-seeking studies—that is, such articles present a new technique, model, or tool to solve a practical problem. We describe their research strategy as *solution-seeking*; all articles that present new solutions also include a knowledge-seeking study to evaluate or validate the solutions. In almost all cases, the evaluation was performed using a Laboratory experiment strategy (we report this as *solution-seeking* + Laboratory Experiment). We note that many such experimental studies are not controlled experiments—instead, solutions are compared to other, existing solutions in order to demonstrate an improvement. Such comparisons aim to collect precise measurements in a contrived setting (so as to maximize the precision of those measurements), which is why we categorize them as laboratory experiments.

Table 10.
Analysis of research strategies in a sample of articles in *Empirical Software Engineering: An International Journal*

Authors	Title	Predominant Strategy and Description	
Springer Empirical Software Engineering Volume 22, Number 1, February 2017			
Luo et al. [124]	FOREPOST: finding performance problems automatically with feedback-directed learning software testing	*Solution-seeking* + Laboratory Experiment	Proposes FOREPOST, a solution for automatically finding performance bottlenecks in applications using black-box software testing. Evaluated in a laboratory experiment.
Vitharana [215]	Defect propagation at the project-level: results and a post-hoc analysis on inspection efficiency	Sample Study	Investigates the impact of defect propagation at the project level during early lifecycle phases. Hypotheses are set up which are tested using a large data set consisting of inspections conducted over a period of 4.5 years at one company.
Niknafs and Berry [148]	The impact of domain knowledge on the effectiveness of requirements engineering activities	Laboratory Experiment	Investigates the impact of domain knowledge on requirements engineering activities. Hypotheses are set up and evaluated through two controlled experiments with students.
Bao et al. [10]	Extracting and analyzing time-series HCI data from screen-captured task videos	Experimental Simulation, *Solution-seeking* + Laboratory Experiment	*Exp. Sim.:* Seeks to better understand challenges in manually transcribing screen-captured videos into time-series HCI data; 3 participants instructed to manually transcribe a 20-min. screen-captured task video. *Solution:* proposes *scvRipper*, a computer-vision based scraping technique to automatically extract time-series HCI data. *lab. exp.:* evaluation of *scvRipper*'s runtime performance and effectiveness.
Li et al. [119]	Zen-ReqOptimizer: a search-based approach for requirements assignment optimization	*Solution-seeking* + Laboratory Experiment	Proposes a fitness function to optimize assignment for reviewing and clarifying requirements to different types of stakeholders. Evaluated in a laboratory setting to compare the performance of 4 other algorithms.

continued on next page

continued from previous page

Authors	Title	Predominant Strategy and Description	
Charpentier et al. [25]	Raters' reliability in clone benchmarks construction	Laboratory Experiment	Investigates the reliability of rater judgments about context-dependent code clones. Evaluated through a laboratory experiment.
Niu et al. [149]	Learning to rank code examples for code search engines	*Solution-seeking* + Laboratory Experiment	Proposes a code example search approach to automatically train a ranking schema that calculates relevance of code examples. Evaluated in a laboratory experiment, demonstrating that the proposed approach outperforms existing ranking schemas.
Ó Cinnéide et al. [28]	An experimental search-based approach to cohesion metric evaluation	*Solution-seeking* + Laboratory Experiment	Proposes a technique to "animate" metrics and observe their behavior; through an experimental evaluation on a set of 10 Java applications with a total of 330KLOC.
Chen and Jiang [26]	Characterizing logging practices in Java-based open source software projects – a replication study in Apache Software Foundation	Sample Study	Replication study to investigate whether or not findings of an earlier study on logging practices in open source systems apply to Apache Software Foundation projects.
Ye et al. [228]	The structure and dynamics of knowledge network in domain-specific Q&A sites: a case study of stack overflow	Field Study	Investigates knowledge diffusion processes in StackOverflow. The 'field' or natural setting is online, because StackOverflow is a Q&A website. Findings are specific to this particular site, but can be used as a foundation to study other, similar sites.
Park et al. [153]	An empirical study of supplementary patches in open source projects	Sample Study	Seeks to understand the characteristics of "multi-fix" bugs and investigate how to predict locations of supplementary patches based on the location of initial patches. Studies a sample of bugs from selected OSS projects, and findings are presented as generalized statements.
Hassan et al. [69]	An empirical study of emergency updates for top android mobile apps	Sample Study	Investigates emergency updates for Android apps, using a sample of 1,000 emergency updates for over 10,000 apps.
Jiang et al. [85]	Why and how developers fork what from whom in GitHub	Sample Study	Investigates why and how developers fork what from whom, using a sample of over 236,000 developers and over 1.8m forks.
Jiang et al. [86]	Do Programmers do Change Impact Analysis in Debugging?	Experimental Simulation, Sample Study	Investigates whether, and how, programmers do change impact analysis. *Exp. Sim.:* recording of 9 hired professional programmers in an environment set up by the researchers; the participants were given bug reports and asked to fix them within an hour. *Sam. Sur.:* online survey with 35 responses.
Springer Empirical Software Engineering Volume 22, Number 2, April 2017			
Kessentini et al. [94]	Search-based detection of model level changes	*Solution-seeking* + Laboratory Experiment	Proposes an approach to detect model changes as a sequence of refactorings. Experimentally evaluated in a contrived setting on models of 8 projects.
Thongtanunam et al. [204]	Review participation in modern code review: An empirical study of the Android, Qt, and OpenStack projects	Sample Study	Studies factors that influence review participation, using a data set of over 196,000 reviews from 3 OSS projects. The study develops a number of models with generalizable statements, and does not focus specifically on the contexts of the 3 projects.

continued on next page

continued from previous page

Authors	Title	Predominant Strategy and Description	
Lokan and Mendes [122]	Investigating the use of moving windows to improve software effort prediction: a replicated study	Sample Study	Data from a sample of 398 projects were used; these projects came from three different organizations, but this is incidental rather than intentional—the focus here is not to capture the context of these three organizations.
Duarte [45]	Productivity paradoxes revisited	Sample Study	Investigates the relationship between quality maturity levels and labor productivity, using a sample data set of 687 firms.
Rojas et al. [165]	A detailed investigation of the effectiveness of whole test suite generation	Laboratory Experiment	Compares different approaches (one goal at a time, whole, archive) to generate test cases, using an experimental study on a random set of 100 Java classes.
Mkaouer et al. [140]	A robust multi-objective approach to balance severity and importance of refactoring opportunities	*Solution-seeking* + Laboratory Experiment	Proposes a multi-objective approach to find the best trade-off between quality improvements, severity, and importance of fixing code smells. The approach is experimentally compared to other approaches.
Kifetew et al. [95]	Generating valid grammar-based test inputs by means of genetic programming and annotated grammars	*Solution-seeking* + Laboratory Experiment	Proposes grammar annotations to generate test data; evaluated through a laboratory experiment which showed that the proposed approach has comparable results to learned probabilities in terms of coverage and fault detection, and achieves a higher level of valid input data.
	Springer Empirical Software Engineering Volume 22, Number 3, June 2017		
Ståhl et al. [195]	Achieving traceability in large scale continuous integration and delivery deployment, usage and validation of the Eiffel framework	*Solution-seeking* + Judgment Study + Field Study	Investigates how to address traceability in large-scale software development in a CI/CD context. Proposes a framework ("Eiffel") which was validated through a judgment study and field study.
Falessi et al. [53]	Estimating the number of remaining links in traceability recovery	*Solution-seeking* + Laboratory Experiment	Proposes an approach to estimate the number of "positive" candidate links that provide traceability. The approach is experimentally evaluated on a data set; performance of multivariate estimation models is compared to univariate models.
Zogaan et al. [233]	Automated training-set creation for software architecture traceability problem	*Solution-seeking* + Laboratory Experiment	Proposes 3 automated techniques to create training sets, which are experimentally evaluated.
Sharif et al. [178]	Eye movements in software traceability link recovery	Experimental Simulation, Judgment Study	Compares the gaze-link method to IR methods. *Exp. Sim.:* One set of participants is asked to perform a number of bug localization tasks *Judgment Study:* A set of participants is asked to rank the results based on utility.
Guo et al. [66]	Tackling the term-mismatch problem in automated trace retrieval	*Solution-seeking* + Laboratory Experiment	Proposes 3 techniques for augmenting queries to generate more accurate trace links. Evaluated through a set of experiments
Behnamghader [15]	A large-scale study of architectural evolution in open-source software systems	Sample Study	Presents an analysis of architectural changes of over 900 versions of 23 OSS systems, using *ARCADE*, which is an approach that the authors published before. (ARCADE enables this knowledge-seeking study).

continued on next page

continued from previous page

Authors	Title	Predominant Strategy and Description	
Wu et al. [226]	Analysis of license inconsistency in large collections of open source projects	Sample Study	Presents an analysis of license inconsistencies on a sample of over 10,500 OSS projects.
Choetkiertikul [27]	Predicting the delay of issues with due dates in software projects	*Solution-seeking* + Sample Study	Presents an approach for project managers to predict whether issues are at risk of delay. Evaluated using a sample of over 108,000 issues from 8 projects.
Coelho et al. [29]	Exception handling bug hazards in Android: Results from a mining study and an exploratory survey	Sample Study (2 ×)	Presents an analysis of exception handling bug hazards faced by app developers. Study 1 is a sample study of over 6,000 Java exception stack traces from over 600 OSS apps. Study 2 is a sample study of over 70 developers.
Munaiah et al. [145]	Do bugs foreshadow vulnerabilities? An in-depth study of the chromium project	Sample Study	Investigates the relationship between bugs and vulnerabilities using a sample of over 374,000 bugs and over 700 post-release vulnerabilities within the Chromium project.
Sawant and Bacchelli [173]	fine-GRAPE: fine-grained APi usage extractor – an approach and dataset to investigate API usage	Sample Study	Investigates usage of 5 APIs by a sample of over 20,000 client projects. The study is facilitated by a purposely developed approach called fine-GRAPE.
Spinellis [194]	A repository of Unix history and evolution	Field Study	Presents an analysis of the history and evolution of Unix. The focus of the study is a single system, which is analyzed through archival data gathered through a repository.
Caneill et al. [22]	The Debsources Dataset: two decades of free and open source software	Field Study	Presents a case study of Debian, which investigates its evolution its rate of change, use and popularity of programming languages, and the use and evolution of licenses within Debian.
Jbara and Feitelson [83]	How programmers read regular code: a controlled experiment using eye tracking	Laboratory Experiment	Seeks to verify and quantify the effect of "regular" code patterns on code comprehension, using a controlled experiment with 18 students and 2 faculty members.
Macleod et al. [126]	Documenting and sharing software knowledge using screencasts	Sample Study	Investigates knowledge sharing through screencasts, by studying a total sample of 27 YouTube videos representing over 8 hours of footage. In addition, interviews with 10 screencast creators were conducted. The study focuses on finding generalizable answers though the authors acknowledge the limitations of the limited sample.
Beller et al. [16]	The last line effect explained	Sample Study	Investigates the *"last line effect,"* the phenomenon that the last line in a 'micro-clone' is more likely to contain an error, through an analysis of a sample of 219 OSS projects, which is complemented with a set of interviews with developers.
Vendome et al. [214]	License usage and changes: a large-scale study on gitHub	2 × Sample Study	Investigates when and why developers adopt or change software licenses, through a quantitative analysis of a sample of over 16,000 Java projects hosted on GitHub. In addition, a qualitative analysis was conducted of a sample of over 1,100 projects was conducted.

Springer Empirical Software Engineering Volume 22, Number 4, 2017

continued on next page

continued from previous page

Authors	Title	Predominant Strategy and Description	
Shi et al. [184]	Metric-based software reliability prediction approach and its application	*Solution-seeking* + Laboratory Experiment + Judgment Study	Proposes a software reliability prediction approach based on metrics. *Lab. exp.:* Experimentally evaluated the effort necessary for the approach. *Jud. Stud.:* a panel of four experts was invited to review the methods and results of the study.
Wu et al. [226]	Assessing the quality of industrial avionics software: an extensive empirical evaluation	Field Study	Presents a case study of an industrial real-time avionics operating system. The analysis focuses on the variation in quality due to testing and associated characteristics in a concrete RTOS.
Li et al. [118]	Which log level should developers choose for a new logging statement?	Sample Study	Investigates log levels through a sample study of over 16,000 logging statements, originating in 4 OSS projects.
Stavropoulou et al. [196]	Case study on which relations to use for clustering-based software architecture recovery	Sample Study	Investigates whether a large software system's architecture can be accurately recovered while minimizing the data sources to do so. Presents the results of an analysis of a small sample of systems to answer a series of research questions.
Assunção et al. [6]	Multi-objective reverse engineering of variability-safe feature models based on code dependencies of system variants	*Solution-seeking* + Sample Study	Proposes an approach for reverse-engineering feature models from feature sets and a dependency graph. The approach is evaluated on a small sample of systems.
Gharehyazie et al. [57]	Tracing distributed collaborative development in apache software foundation projects	2 × Sample Study	Presents an algorithm for tracing group collaborations in OSS, which was then used to study associations of teams.
Li et al. [119]	Towards just-in-time suggestions for log changes	Sample Study	The study identified all log statements in 4 OSS projects, and identified 20 different reasons for log changes. The goal here was to learn characteristics about the population, not about the specific OSS projects.
Herbold et al. [71]	Global vs. local models for cross-project defect prediction	Laboratory Experiment	Presents a replication study of cross-project defect prediction, which evaluates the performance of local models, followed by a comparison of a global model.
Menzies et al. [137]	Are delayed issues harder to resolve? Revisiting cost-to-fix of defects throughout the lifecycle	Sample Study	Investigates the *"delayed issue"* effect, which states that the longer an issue lingers, the more effort required to resolve it, through a study involving a sample of 171 projects.
Martinez et al. [129]	Automatic repair of real bugs in java: a large-scale experiment on the Defects4j dataset	Laboratory Experiment	Investigates the effectiveness of automatic test-suite based repair on Defects4J, which is a large data set of real-world Java bugs. Using state-of-the-art repair methods, it was demonstrated that patches could be generated for 47 out of 224 bugs.
Mahmoud and Bradshaw [127]	Semantic topic models for source code analysis	*Solution-seeking* + Laboratory Experiment	Proposes an approach for topic modeling that is designed for source code, which is experimentally evaluated.
Sakhnini et al. [172]	Group versus individual use of power-only EPMcreate as a creativity enhancement technique for requirements elicitation	Laboratory Experiment	Presents an experiment to evaluate how the size of a group that uses the *Power-Only EPMcreate* (POEPMcreate) creativity technique affects the effectiveness of both the group and each group member, in terms of generating requirement ideas.

continued on next page

continued from previous page

Authors	Title	Predominant Strategy and Description	
Joblin et al. [87]	Evolutionary trends of developer coordination: a network approach	Sample Study	Investigates the evolution of developer coordination in open source projects, through an analysis of a sample of 18 large OSS projects, with findings that aim to generalize.
Lin et al. [120]	Studying the urgent updates of popular games on the Steam platform	Sample Study	Investigates urgent updates of the 50 most popular games on the Steam platform, in particular, how often developers release urgent updates, and why.
Riaz et al. [163]	Identifying the implied: Findings from three differentiated replications on the use of security requirements templates	Laboratory Experiment	Presents 3 replications of a controlled experiment to evaluate the use of automatically-suggested templates in identifying implicit security requirements versus a manual approach.
Lenberg et al. [115]	An initial analysis of software engineers' attitudes towards organizational change	Sample Study	Creates, verifies, and validates a model that predicts software engineers' attitudes towards organizational change, using sample data collected through a questionnaire, with 56 responses.
Johanson and Hasselbring [88]	Effectiveness and efficiency of a domain-specific language for high-performance marine ecosystem simulation: a controlled experiment	Experimental Simulation	The scientist participants' workflow is "imitated"; the context is fairly realistic, using a concrete system (the Sprat Ecosystem DSL).
Le et al. [112]	Will this localization tool be effective for this bug? Mitigating the impact of unreliability of information retrieval based bug localization tools	*Solution-seeking* + Laboratory Experiment	Presents an oracle that can predict whether a ranked list produced by an information retrieval-based bug localization tool is likely to be effective or not. Evaluated on a data set of over 3,000 bug reports.
	Springer Empirical Software Engineering Volume 22, Number 5, December 2017		
Méndez Fernández et al. [55]	Naming the pain in requirements engineering	Sample Study	Presents results of a sample survey (part of the NaPiRE initiative) with data from 228 companies on problems in requirements engineering.
Bano et al. [9]	User satisfaction and system success: an empirical exploration of user involvement in software development	Field Study	Presents a longitudinal case study of a software development project to explore user satisfaction in relation to user involvement and system success.
Lehtinen et al. [113]	Recurring opinions or productive improvements—what agile teams actually discuss in retrospectives	Field Study	Presents a longitudinal case study based on data collected from 37 team-level retrospective meetings, within the context of a software development company of 800 employees.
Dieste et al. [41]	Empirical evaluation of the effects of experience on code quality and programmer productivity: an exploratory study	Laboratory Experiment	Investigates the effects of experience on code quality and programmer productivity. Presents results of 10 quasi-experiments conducted in academic and industry settings (in all cases the experiments were conducted in contrived settings).
Jongeling et al. [89]	On negative results when using sentiment analysis tools for software engineering research	Laboratory Experiment	Investigates to what extent results from SE studies using sentiment analysis depend on the choice of sentiment analysis tool. Through experimental comparison, finds that tools do not compare with manual labeling, nor do different tools agree with each other.
Gil and Lalouche [58]	On the correlation between size and metric validity	Sample Study	Presents an analysis of a set of 26 metrics and a data set of over 53,000 Java source code files that demonstrates that the validity of metrics depends on their correlation with size.

continued on next page

continued from previous page

Authors	Title	Predominant Strategy and Description	
Sabané et al. [171]	Fragile base-class problem, problem?	2 × Sample Study	Sample Study 1 presents an analysis of over 112,000 "micro-architectures" (called Fragile Base-Class Problem, FBCP); Sample Study 2 presents an analysis of 41 responses.
Menzies et al. [138]	Negative results for software effort estimation	Laboratory Experiment	Experimental study that investigates whether new software development effort estimation techniques generate better estimates than older methods.
King et al. [96]	To log, or not to log: using heuristics to identify mandatory log events – a controlled experiment	Laboratory Experiment	Presents a controlled experiment with over 100 students to evaluate the use of a heuristics-driven method for identifying mandatory log events.
	Springer Empirical Software Engineering Volume 22, Number 6, December 2017		
Palomares et al. [152]	Requirements reuse and requirement patterns: a state of the practice survey	Sample Study	Presents a survey to investigate the state of practice of reuse of requirements; results include an analysis of 71 responses from requirements engineers.
Tosun et al. [209]	An industry experiment on the effects of test-driven development on external quality and productivity	Laboratory Experiment	The setting is contrived—developers had an opportunity to sign up to participate; the research setting exists solely for the purpose of this experiment.
Malhotra and Khanna [128]	An empirical study for software change prediction using imbalanced data	Laboratory Experiment	Develops a set of change prediction models, and experimentally evaluates these on imbalanced software data sets.
Heikkilä et al. [70]	Managing the requirements flow from strategy to release in large-scale agile development: a case study at Ericsson	Field Study	Investigates the "flow" of requirements from strategy to system release in a large-scale agile context in Ericsson.
Alégroth and Feldt [3]	On the long-term use of visual GUI testing (VGT) in industrial practice: a case study	Field Study	Presents a study that evaluates the use of visual GUI testing at Spotify, which address questions regarding adoption of VGT, its benefits, and challenges.
Labunets et al. [108]	Model comprehension for security risk assessment: an empirical comparison of tabular vs. graphical representations	Laboratory Experiment	Presents results of a series of experiments to evaluate the effectiveness of tabular and graphical representations for extracting information about security risks.
Antinyan et al. [4]	Evaluating code complexity triggers, use of complexity measures and the influence of code complexity on maintenance time	Sample Study	Presents a sample survey that investigates code characteristics and how they contribute to complexity, based on a set of 100 responses.
Noei et al. [150]	A study of the relation of mobile device attributes with the user-perceived quality of Android apps	Sample Study	Investigates the relationship between device attributes and app attributes on the one hand, and the user-perceived quality of apps on the other. The study is based on an analysis of over 150,000 reviews, 30 devices, and 280 apps.
Bezemer et al. [18]	An empirical study of unspecified dependencies in make-based build systems	Field Study	Presents an analysis of 4 OSS systems to identify unspecified dependencies in make-based build systems.
Xia et al. [227]	What do developers search for on the web?	Sample Study	Investigates what developers search for on the web, through a set of queries from 60 developers and a survey among 235 software engineers.

continued on next page

continued from previous page

Authors	Title	Predominant Strategy and Description	
Munaiah et al. [145]	Curating GitHub for engineered software projects	Laboratory Experiment	Presents a framework and reference implementation of a tool ("reaper") that enables researchers to identify GitHub repositories used for software development (instead of other purposes). The tool is experimentally evaluated using a set of GitHub projects. The focus is on the evaluation of the tool, rather than to establish knowledge about the sample of projects.

B GUIDELINES FOR USING THE ABC FRAMEWORK TO DESIGN A RESEARCH PROGRAM

One of the ways to use the ABC framework is to design a research program. The ABC framework suggests that any research strategy ultimately comes down to a trade-off between generalizability of findings to a population of actors (A), precision of measurement of behavior (B), and realism of context (C). Each research strategy has certain inherent limitations, and potential strengths that it can achieve. By combining studies with different research strategies, weaknesses inherent in one study may be ameliorated by another study with a different strategy.

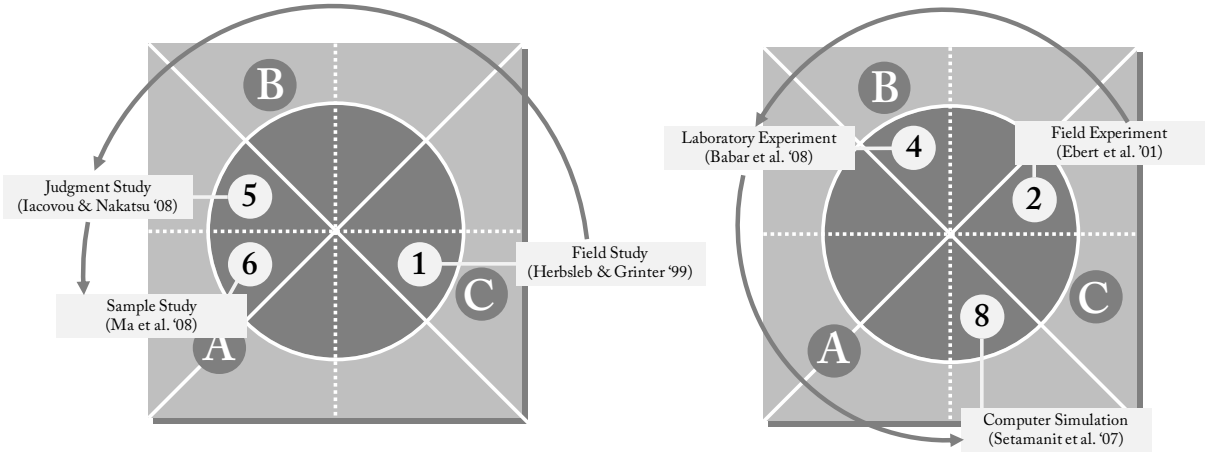
While norms and customs vary across the world in terms of the requirements for a PhD dissertation, it is custom in several countries in Europe (including Sweden, Finland, and the Netherlands) to conduct a series of studies, which will be written up either as a monograph or a set of published papers. Our examples here assume that a student will undertake three studies (though obviously this number may vary depending on norms in their particular institution). By way of guidance, we develop two hypothetical scenarios, drawing on the studies presented in Sec. 4.1 on Global Software Engineering. In each of these scenarios, we imagine how a PhD student could design a research program by conducting a series of studies using different research strategies. Fig. 2 shows these in schematic form. We discuss each scenario in more detail below.

B.1 Scenario I: Exploring and Understanding

The research program of Scenario I consists of the studies listed in Table 11. All three studies focus on challenges of distributed development; the field study ① documents the various challenges involved in integration of software components. The Judgment Study ⑤ explores the risk factors for offshore-outsourcing that managers face. The Sample Study ⑥, finally, explores three key issues in software development faced by an offshore software supplier. In this scenario, the topic of study is quite new and relatively unexplored. This has parallels with what Edmondson and McManus [50] label as the *nascent* research stage for a particular topic. Thus, research is about exploring and understanding the topic better. Through a Field Study a high level of realism of context can be achieved; and through the Sample Study a high degree of generalizability of findings can be achieved. The Judgment Study is a compromise between achieving a high level of precision and generalizability of findings, not quite achieving a maximum potential of either.

B.2 Scenario II: Measuring and Testing

In the second scenario, the research program consists of the three studies listed in Table 12. These studies are positioned in three of the four quadrants of the ABC framework (see Fig. 2). Each of these studies investigates the impact of certain aspects of distributed development (e.g., co-location vs. distributed work; task allocation) on the three key concerns that underpin the "software crisis," namely software quality, cost, and project duration. Because each study has a different strategy, each can potentially leverage inherent strengths to overcome particular weaknesses. For example, the field experiment ② benefits from a realistic setting, but this negatively affects the precision of measurement. The laboratory experiment ④, on the other hand, can achieve a high precision



Scenario I: Exploring and Understanding

Scenario II: Measuring and Testing

Fig. 2. Two hypothetical scenarios for designing a research program. Numbers refer to the studies in Table 6, e.g. ① refers to the first study, which is a Field Study by Herbsleb and Grinter. The labels A, B, and C indicate, respectively, the maximum potential for generalizability of findings, maximum potential for precision of measurement of behavior, and maximum potential to capture a realistic context.

Table 11. Research program of Scenario I

Study	Research goal	Strategy	Setting	Strengths	Weaknesses
① Herbsleb & Grinter [73]	To develop an understanding of the difficulties of distributed software development.	Field Study	Natural setting	Highly realistic context facilitating a deep understanding of the GSE phenomenon from one company's perspective.	No statistically generalizable findings. Low level of precision of measurement of behavior: no causal relationships can be established.
⑤ Iacovou & Nakatsu [78]	To investigate risk factors for offshore-outsourcing software development.	Judgment Study	Neutral setting	Allowed for a relatively high level of control to precisely capture the various risk factors from a panel of experts.	Results lack the realism found in a specific concrete setting. Participants not a representative sample (rather, a systematic sample).
⑥ Ma et al. [125]	Investigation of key issues in software development by Chinese software suppliers: language barriers, channels of communication, and working overtime.	Sample Study	Neutral setting	Sample allowing generalizability of findings to a large population.	The role of context is minimized, because all questions in the survey must be answerable by all respondents, each of which has a unique context.

of measurement, but suffers from a low level of realism. The third study ⑧, finally, can overcome the cost of setting up laboratory experiments and field experiments, by running a large number of scenarios with a computer simulation. However, the computer simulation also suffers from a lack of realism, and in fact, this strategy only results in theoretical findings, not empirical findings. Together, however, these three studies can form a

research program for a PhD dissertation focused on investigating how global software engineering can address the traditional concerns of the software crisis.

The focus of Scenario II is much more on measuring and testing, whereas Scenario I was more about exploring and understanding the topic. In Scenario II, there is more structure for the topic in that constructs have emerged that should be the focus of measurement and testing. This corresponds with the *intermediate* stage of a research topic as defined by Edmondson and McManus [50].

The studies in Scenario II vary in research setting, however, while some realism has been captured with the Field Experiment, and a high level of precision of measurement could be achieved through the Laboratory Experiment, none of the three studies achieve generalizability of findings.

We could also define a third scenario which we could label *Theorising and Bounding*. This involves greater conceptualisation, and theorising about the relationships between concepts. There is also sufficient knowledge from prior research to be more confident about defining precise constructs for formal experimentation. This corresponds with the *mature* stage of a research topic as defined by Edmondson and McManus [50].

Table 12. Research program of Scenario II

Study	Research goal	Strategy	Setting	Strengths	Weaknesses
② Ebert et al. [49]	To investigate the impact of co-location, coaching, and teamwork on software quality and costs.	Field Experiment	Natural setting	Company setting offers maximum level of realism of study context.	Results may be strongly linked to the specific organization due to confounding variables that cannot be isolated in a natural setting.
④ Babar et al. [8]	To investigate the impact of groupware support for distributed teams on the quality of architecture evaluation deliverables.	Laboratory Experiment	Contrived setting	High precision of measurement to establish internal validity.	Low level of realism due to: contrived setting; use of undergrad students; task limited to one task only. Limited level of generalizability of findings.
⑧ Setamanit et al. [177]	To investigate task allocation strategies in GSE and its impact on project duration.	Computer Simulation	Non-empirical setting	Ability to run a large number of complex scenarios that are hard or too costly to evaluate in real-world settings.	Lack of realism as the simulation depends on rules that must be preprogrammed. No new empirical results are observed.

REFERENCES

[1] S. Adolph, P. Kruchten, and W. Hall. 2012. Reconciling perspectives: A grounded theory of how people manage the process of software development. *Journal of Systems and Software* 85 (2012), 1269–1286.

[2] P.J. Ågerfalk and B. Fitzgerald. 2006. Flexible and Distributed Software Processes: Old Petunias in New Bowls? *Commun ACM* 49, 10 (2006), 27–34.

[3] E. Alégroth and R. Feldt. 2017. On the long-term use of visual GUI testing in industrial practice: a case study. *Empir Software Eng* 22, 6 (2017), 2937–2971.

[4] V. Antinyan, M. Staron, and A. Sandberg. 2017. Evaluating code complexity triggers, use of complexity measures and the influence of code complexity on maintenance time. *Empir Software Eng* 22, 6 (2017), 3057–3087.

[5] E. Arisholm, H. Gallis, T. Dybå, and D.I.K. Sjøberg. 2007. Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise. *IEEE Trans. Softw. Eng.* 33, 2 (2007), 65–86.

[6] W. K. G. Assunção, R. E. Lopez-Herrejon, L. Linsbauer, S. R. Vergilio, and A. Egyed. 2017. Multi-objective reverse engineering of variability-safe feature models based on code dependencies of system variants. *Empir Software Eng* 22, 4 (2017), 1763–1794.

[7] D. Avison, F. Lau, M. Myers, and P.A. Nielsen. 1999. Action Research. *Commun ACM* 42, 1 (1999), 94–97.

[8] M.A. Babar, B. Kitchenham, and R. Jeffery. 2008. Comparing distributed and face-to-face meetings for software architecture evaluation: A controlled experiment. *Empir Software Eng* 13, 1 (2008), 39–62.

[9] M. Bano, D. Zowghi, and F. da Rimini. 2017. User satisfaction and system success: an empirical exploration of user involvement in software development. *Empir Software Eng* 22, 5 (2017), 2339–2372.

[10] L. Bao, J. Li, Z. Xing, X. Wang, X. Xia, and B. Zhou. 2017. Extracting and analyzing time-series HCI data from screen-captured task videos. *Empir Software Eng* 22, 1 (2017), 134–174.
[11] V.R. Basili, R.W. Selby, and D.H. Hutchens. 1986. Experimentation in Software Engineering. *IEEE Trans Softw Eng* 12, 7 (1986), 733–743.
[12] V.R. Basili and M.V. Zelkowitz. 2007. Empirical studies to build a science of computer science. *Commun ACM* 50, 11 (2007), 33–37.
[13] B.C.D. Anda, D.I.K. Sjøberg, and A. Mockus. 2009. Variability and Reproducibility in Software Engineering: A Study of Four Companies that Developed the Same System. *IEEE Trans. Softw. Eng.* 35, 3 (2009), 407–429.
[14] S. Beecham, N. Baddoo, T. Hall, H. Robinson, and H. Sharp. 2008. Motivation in Software Engineering: A systematic literature review. *Inform Software Tech* 50, 9-10 (2008), 860–878.
[15] P. Behnamghader, D. M. Le, J. Garcia, D. Link, A. Shahbazian, and N. Medvidovic. 2017. A large-scale study of architectural evolution in open-source software systems. *Empir Software Eng* 22, 3 (2017), 1146–1193.
[16] M. Beller, A. Zaidman, A. Karpov, and R.A. Zwaan. 2017. The last line effect explained. *Empir Software Eng* 22, 3 (2017), 1508–1536.
[17] D.M. Berry and W.F. Tichy. 2003. Comments on "Formal Methods Application: An Empirical Tale of Software Development". *IEEE Trans Softw Eng* 29, 6 (2003).
[18] C.-P. Bezemer, S. McIntosh, B. Adams, D. M. German, and A. E. Hassan. 2017. An empirical study of unspecified dependencies in make-based build systems. *Empir Software Eng* 22, 6 (2017), 3117–3148.
[19] N. Bos, N. Sadat Shami, J.S. Olson, A. Cheshin, and N. Nan. 2004. In-group/Out-group Effects in Distributed Teams: An Experimental Simulation. In *Proc. International Conference on Computer-Supported Cooperative Work and Social Computing (CSCW)* (New York, NY, USA). ACM, 429–436.
[20] S.S. Brilliant and J.C. Knight. 1999. Empirical research in software engineering: a workshop. *ACM SIGSOFT Software Engineering Notes* 24, 3 (1999), 44–52.
[21] A. Bryant. 2000. 'It's Engineering Jim ... but not as we know it': Software Engineering - solution to the software crisis, or part of the problem?. In *Proc. International Conference on Software Engineering* (Limerick, Ireland). 77–86.
[22] M. Caneill, D.M. Germán, and S. Zacchiroli. 2017. The Debsources Dataset: two decades of free and open source software. *Empir Software Eng* 22, 3 (2017), 1405–1437.
[23] E. Capra, C. Francalanci, and F. Merlo. 2008. An Empirical Study on the Relationship among Software Design Quality, Development Effort, and Governance in Open Source Projects. *IEEE Trans Softw Eng* 34, 6 (2008).
[24] E. Carmel. 1999. *Global Software Teams.* Prentice Hall.
[25] A. Charpentier, J.-R. Falleri, F. Morandat, E. Ben Hadj Yahia, and L. Réveillère. 2017. Raters' reliability in clone benchmarks construction. *Empir Software Eng* 22, 1 (2017), 235–258.
[26] B. Chen and Z.M. Jiang. 2017. Characterizing logging practices in Java-based open source software projects – a replication study in Apache Software Foundation. *Empir Software Eng* 22, 1 (2017), 330–374.
[27] M. Choetkiertikul, H. K. Dam, T. Tran, and A. Ghose. 2017. Predicting the delay of issues with due dates in software projects. *Empir Software Eng* 22, 3 (2017), 1223–1263.
[28] M. Ó Cinnéide, I. Hemati Moghadam, M. Harman, S. Counsell, and L. Tratt. 2017. An experimental search-based approach to cohesion metric evaluation. *Empir Software Eng* 22, 1 (2017), 292–329.
[29] R. Coelho, L. Almeida, G. Gousios, A. van Deursen, and C. Treude. 2017. Exception handling bug hazards in Android. *Empir Software Eng* 22, 3 (2017), 1264–1304.
[30] K. Conboy and B. Fitzgerald. 2010. Method and Developer Characteristics for Effective Agile Method Tailoring: A Study of XP Expert Opinion. *ACM Trans. Softw. Eng. Methodol.* 20, 1 (2010).
[31] D.S. Cruzes and T. Dybå. 2011. Research Synthesis in Software Engineering: A Tertiary Study. *Inform Software Tech* 53 (2011), 440–455.
[32] B. Curtis. 1980. Measurement and experimentation in software engineering. *Proc. IEEE* 68, 9 (1980), 1144–1157.
[33] B. Curtis. 1984. Fifteen Years of Psychology in Software Engineering: Individual Differences and Cognitive Science. In *Proc. 7th International Conference on Software Engineering (ICSE '84).* IEEE Press, Piscataway, NJ, USA, 97–106.
[34] B. Curtis. 2009. Point/Counterpoint: Are Rigorous Experiments Realistic for Software Engineering? *IEEE Softw* 26, 6 (2009), 56–59.
[35] B. Curtis, E.M. Soloway, R.E. Brooks, J.B. Black, K. Ehrlich, and H.R. Ramsey. 1986. Software Psychology: The Need for an Interdisciplinary Program. *Proc IEEE* 74, 8 (1986), 1092–1106.
[36] N. Dalkey and O. Helmer. 1963. An Experimental Application of the Delphi Method to the Use of Experts. *Manage Sci* 9, 3 (1963), 458–467.
[37] D.E. Damian and D. Zowghi. 2003. RE challenges in multi-site software development organisations. *Requir Eng* 8 (2003), 149–160.
[38] M. Daneva. 2010. Engineering the Coordination Requirements in Cross-organizational ERP Projects. *Enterprise Information Systems and Implementing IT Infrastructures* (2010), 1–19.
[39] M. Daneva and N. Ahituv. 2010. A Focus Group Study on Inter-organizational ERP Requirements Engineering Practices. In *Proc. 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement.* ACM.
[40] C. De Souza, Y. Dittrich, H. Sharp, and J. Singer. 2009. Cooperative and Human Aspects of Software Engineering (CHASE 2009). In *Proc. International Conference on Software Engineering (Companion Volume).* 451–452.

[41] O. Dieste, A. M. Aranda, F. Uyaguari, B. Turhan, A. Tosun, D. Fucci, M. Oivo, and N. Juristo. 2017. Empirical evaluation of the effects of experience on code quality and programmer productivity: an exploratory study. *Empir Software Eng* 22, 5 (2017), 2457–2542.
[42] Y. Dittrich. 2000. Beg, Borrow, and Steal — But What, and What For?. In *Workshop: Beg, Borrow, or Steal: Using Multidisciplinary Approaches in Empirical Software Engineering Research (co-located with ICSE'00)* (Limerick, Ireland).
[43] Y. Dittrich, M. John, J. Singer, and B. Tessem. 2007. For the Special issue on Qualitative Software Engineering Research. *Inform Software Tech* 49, 6 (2007).
[44] L. Dobrica and E. Niemelä. 2005. A Survey on Software Architecture Analysis Methods. *IEEE Trans Softw Eng* 28, 7 (2005), 638–653.
[45] C.H. C. Duarte. 2017. Productivity paradoxes revisited. *Empir Software Eng* 22, 2 (2017), 818–847.
[46] T. Dybå, R. Prikladnicki, K. Rönkkö, C. Seaman, and J. Sillito. 2011. Qualitative research in software engineering. *Empir Software Eng* 16 (2011), 425–429.
[47] T. Dybå, D.I.K. Sjøberg, and D.S. Cruzes. 2012. What Works for Whom, Where, When, and Why? On the Role of Context in Empirical Software Engineering. In *Proc. International Symposium on Empirical Software Engineering and Measurement (ESEM)* (Lund, Sweden). ACM.
[48] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian. 2008. Selecting Empirical Methods for Software Engineering Research. In *Guide to Advanced Software Engineering*, Forest Shull, Janice Singer, and Dag I.K. Sjøberg (Eds.).
[49] C. Ebert, C.H. Parro, R. Suttels, and H. Kolarczyk. 2001. Better validation in a world-wide development environment. In *Proc. 7th International Software Metrics Symposium (METRICS)*.
[50] A.C. Edmondson and S.E. McManus. 2007. Methodological Fit in Management Field Research. *Acad. Manage. Rev.* 32, 4 (2007).
[51] H. Edwards, S. McDonald, and M. Young. 2009. The repertory grid technique: Its place in empirical software engineering research. *Inform Software Tech* 51, 4 (2009), 785–798.
[52] J.A. Espinosa and E. Carmel. 2003. The Impact of Time Separation on Coordination in Global Software Teams: a Conceptual Foundation. *Softw. Process Improve. Pract.* 8, 4 (2003), 249–266.
[53] D. Falessi, M. Di Penta, G. Canfora, and G. Cantone. 2017. Estimating the number of remaining links in traceability recovery. *Empir Software Eng* 22, 3 (2017), 996—1027.
[54] N. Fenton and S. L. Pfleeger. 1997. *Software Metrics: A Rigorous and Practical Approach* (2nd (revised printing) ed.). PWS Publishing Company.
[55] D. Méndez Fernández, S. Wagner, M. Kalinowski, M. Felderer, P. Mafra, A. Vetrò, T. Conte, M.-T. Christiansson, D. Greer, C. Lassenius, T. Männistö, M. Nayabi, M. Oivo, B. Penzenstadler, D. Pfahl, R. Prikladnicki, G. Ruhe, A. Schekelmann, S. Sen, R. Spinola, A. Tuzcu, J. L. de la Vara, and R. Wieringa. 2017. Naming the pain in requirements engineering. *Empir Software Eng* 22, 5 (2017), 2298–2338.
[56] B. Fitzgerald and D. Howcroft. 1998. Towards dissolution of the IS research debate: from polarization to polarity. *J Inform Technol* 13, 4 (1998), 313–326.
[57] M. Gharehyazie and V. Filkov. 2017. Tracing distributed collaborative development in apache software foundation projects. *Empir Software Eng* 22, 4 (2017), 1795–1830.
[58] Y. Gil and G. Lalouche. 2017. On the correlation between size and metric validity. *Empir Software Eng* 22, 5 (2017), 2585–2611.
[59] B.G. Glaser. 1978. *Theoretical Sensitivity.* The Sociology Press.
[60] B.G. Glaser and A.L. Strauss. 1967. *The Discovery of Grounded Theory.* AldineTransaction.
[61] R.L. Glass. 1994. The Software-Research Crisis. *IEEE Softw* 11, 6 (1994).
[62] R.L. Glass. 2002. *Facts and Fallacies of Software Engineering.* Addison Wesley.
[63] R.L. Glass, I. Vessey, and V. Ramesh. 2002. Research in software engineering: an analysis of the literature. *Inform Software Tech* 44 (2002).
[64] D. Graziotin, X. Wang, and P. Abrahamsson. 2014. Happy software developers solve problems better: psychological measurements in empirical software engineering. *PeerJ* 2, e289 (2014).
[65] S. Gregor. 2006. The nature of theory in information systems. *MIS Quart* 30, 3 (2006), 611–642.
[66] J. Guo, M. Gibiec, and J. Cleland-Huang. 2017. Tackling the term-mismatch problem in automated trace retrieval. *Empir Software Eng* 22, 3 (nov 2017), 1103–1142.
[67] J.E. Hannay and M. Jørgensen. 2008. The Role of Deliberate Artificial Design Elements in Software Engineering Experiments. *IEEE Trans Softw Eng* 34, 2 (2008).
[68] R. Harrison, N. Badoo, E. Barry, S. Biffl, A. Parra, B. Winter, and J. Wuest. 1999. Directions and Methodologies for Empirical Software Engineering Research. *Empir Softw Eng* 4, 4 (1999), 405–410.
[69] S. Hassan, W. Shang, and A.E. Hassan. 2017. An empirical study of emergency updates for top android mobile apps. *Empir Software Eng* 22, 1 (2017), 505–546.
[70] V.T. Heikkilä, M. Paasivaara, C. Lasssenius, D. Damian, and C. Engblom. 2017. Managing the requirements flow from strategy to release in large-scale agile development: a case study at Ericsson. *Empir Software Eng* 22, 6 (2017), 2892–2936.
[71] S. Herbold, A. Trautsch, and J. Grabowski. 2017. Global vs. local models for cross-project defect prediction. *Empir Software Eng* 22, 4 (2017), 1866–1902.

[72] J.D. Herbsleb and R.E. Grinter. 1999. Architectures, Coordination, and Distance: Conway's Law and Beyond. *IEEE Softw* 16, 5 (1999), 63–70.
[73] J.D. Herbsleb and R.E. Grinter. 1999. Splitting the Organization and Integrating the Code: Conway's Law Revisited. In *Proc. International Conference on Software Engineering* (Los Angeles, CA). 85–95.
[74] R. Hoda, J. Noble, and S. Marshall. 2013. Self-Organizing Roles on Agile Software Development Teams. *IEEE Trans Softw Eng* 39, 3 (2013), 422–444.
[75] A. Höfer and W. Tichy. 2007. Status of Empirical Research in Software Engineering. In *Empirical Software Engineering Issues, LNCS 4336*. 10–19.
[76] G. Hofstede, B. Neuijen, D.D. Ohayv, and G. Sanders. 1990. Measuring Organizational Cultures: A Qualitative and Quantitative Study Across Twenty Cases. *Administrative Science Quarterly* 35, 2 (1990).
[77] M. Höst, B. Regnell, J.N. och Dag, J. Nedstam, and C. Nyberg. 2001. Exploring bottlenecks in market-driven requirements management processes with discrete event simulation. *J Sys Softw* 59, 3 (2001), 323–332.
[78] C.L. Iacovou and R. Nakatsu. 2008. A risk profile of offshore-outsourced development projects. *Commun ACM* 51, 6 (2008), 89–94.
[79] M. Ivarsson and T. Gorschek. 2011. A method for evaluating rigor and industrial relevance of technology evaluations. *Empir Softw Eng* 16, 3 (2011), 365–395.
[80] Tisseau J. 2008. In vivo, in vitro, in silico, in virtuo. In *1st Workshop on SMA in Biology at meso or macroscopic scales* (Paris).
[81] S. Jain, M.A. Babar, and J. Fernandez. 2013. Conducting Empirical Studies in Industry: Balancing Rigor and Relevance. In *Proc. International Workshop on Conducting Empirical Studies in Industry (CESI)*.
[82] H. Jansen. 2010. The Logic of Qualitative Survey Research and its Position in the Field of Social Research Methods. *Forum: Qualitative Social Research* 11, 2 (2010).
[83] A. Jbara and D.G. Feitelson. 2017. How programmers read regular code: a controlled experiment using eye tracking. *Empir Software Eng* 22, 3 (2017), 1440–1477.
[84] D.R. Jeffery and L.G. Votta. 1999. Guest Editor's Special Section Introduction. *IEEE Trans Softw Eng* 25, 4 (1999), 435–437.
[85] J. Jiang, D. Lo, J. He, X. Xia, P.S. Kochhar, and L. Zhang. 2017. Why and how developers fork what from whom in GitHub. *Empir Software Eng* 22, 1 (2017), 547–578.
[86] S. Jiang, C. McMillan, and R. Santelices. 2017. Do Programmers do Change Impact Analysis in Debugging? *Empir Software Eng* 22, 2 (2017), 631–669.
[87] M. Joblin, S. Apel, and W. Mauerer. 2017. Evolutionary trends of developer coordination: a network approach. *Empir Software Eng* 22, 4 (2017), 2050–2094.
[88] A. N. Johanson and W. Hasselbring. 2017. Effectiveness and efficiency of a domain-specific language for high-performance marine ecosystem simulation: a controlled experiment. *Empir Software Eng* 22, 4 (2017), 2206–2236.
[89] R. Jongeling, P. Sarkar, S. Datta, and A. Serebrenik. 2017. On negative results when using sentiment analysis tools for software engineering research. *Empir Software Eng* 22, 5 (2017), 2543–2584.
[90] H. Jordan, S. Beecham, and G. Botterweck. 2014. Modelling Software Engineering Research with RSML. In *Proc. 18th International Conference on Evaluation and Assessment in Software Engineering*.
[91] N. Juristo and A.M. Moreno. 2001. *Basics of Software Engineering Experimentation.* Springer Science+Business Media, LLC.
[92] E Kalliamvakou, G Gousios, K Blincoe, L Singer, DM German, and D Damian. 2014. The Promises and Perils of Mining GitHub. In *Proc. 11th Working Conference on Mining Software Repositories*.
[93] R. Kazman, M. Klein, and P. Clements. 2000. ATAM: Method for Architecture Evaluation. (2000). Carnegie Mellon Software Engineering Institute, Technical Report CMU/SEI-2000-TR-004.
[94] M. Kessentini, U. Mansoor, M. Wimmer, A. Ouni, and K. Deb. 2017. Search-based detection of model level changes. *Empir Software Eng* 22, 2 (2017), 670–715.
[95] F. M. Kifetew, R. Tiella, and P. Tonella. 2017. Generating valid grammar-based test inputs by means of genetic programming and annotated grammars. *Empir Software Eng* 22, 2 (2017), 928–961.
[96] J. King, J. Stallings, M. Riaz, and L. Williams. 2017. To log, or not to log: using heuristics to identify mandatory log events – a controlled experiment. *Empir Software Eng* 22, 5 (2017), 2684–2717.
[97] B.A. Kitchenham and S.L. Pfleeger. 2002. Principles of Survey Research Part 2: Designing a Survey. *ACM Software Engineering Notes* 27, 1 (2002).
[98] B.A. Kitchenham and S.L. Pfleeger. 2002. Principles of Survey Research Part 3: Constructing a Survey Instrument. *ACM Software Engineering Notes* 27, 2 (2002).
[99] B.A. Kitchenham and S.L. Pfleeger. 2002. Principles of Survey Research Part 4: Questionnaire Evaluation. *ACM Software Engineering Notes* 27, 3 (2002).
[100] B.A. Kitchenham and S.L. Pfleeger. 2002. Principles of Survey Research Part 5: Populations and Samples. *ACM Software Engineering Notes* 27, 5 (2002).

[101] B.A. Kitchenham and S.L. Pfleeger. 2003. Principles of Survey Research Part 6: Data Analysis. *ACM Software Engineering Notes* 28, 2 (2003).
[102] B.A. Kitchenham, S.L. Pfleeger, L.M.P Pickard, P.W. Jones, D.C. Hoaglin, K. El Emam, and J. Rosenberg. 2008. Preliminary Guidelines for Empirical Research in Software Engineering. *IEEE Trans. Softw. Eng.* 28, 2 (2008), 721–734.
[103] A.J. Ko, B.A. Myers, M.J. Coblenz, and H.H. Aung. 2006. An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information during Software Maintenance Tasks. *IEEE Trans Softw Eng* 32, 12 (2006).
[104] J. Kontio, J. Bragge, and L. Lehtola. 2008. The Focus Group Method as an Empirical Tool in Software Engineering. In *Guide to Advanced Empirical Software Engineering*. Springer.
[105] M. Krafft, K. Stol, and B. Fitzgerald. 2016. How Do Free/Open Source Developers Pick Their Tools? A Delphi Study of the Debian Project. In *Proc. 38th International Conference on Software Engineering (Companion Volume)* (Austin, TX, USA). 232–241.
[106] P. Kruchten, R.L. Nord, and I. Ozkaya. 2012. Technical Debt: From Metaphor to Theory and Practice. *IEEE Softw* 29, 6 (2012), 18–21.
[107] I. Kwan, A. Schröter, and D. Damian. 2011. Does Socio-Technical Congruence Have an Effect on Software Build Success? A Study of Coordination in a Software Project. *IEEE Trans Softw Eng* 37, 3 (2011).
[108] K. Labunets, F. Massacci, F. Paci, S. Marczak, and F. M. de Oliveira. 2017. Model comprehension for security risk assessment: an empirical comparison of tabular vs. graphical representations. *Empir Software Eng* 22, 6 (2017), 3017–3056.
[109] F. Lanubile. 1997. Empirical Evaluation of Software Maintenance Technologies. *Empir Softw Eng* 2 (1997), 97–108.
[110] T.D. LaToza, M. Chen, L. Jiang, M. Zhao, and A. van der Hoek. 2015. Borrowing from the Crowd: A Study of Recombination in Software Design Competitions. In *Proc. International Conf. Software Engineering*.
[111] S. Lauesen and O. Vinter. 2001. Preventing Requirement Defects: An Experiment in Process Improvement. *Requir Eng* 6 (2001), 37–50.
[112] T.-D. B. Le, F. Thung, and D. Lo. 2017. Will this localization tool be effective for this bug? Mitigating the impact of unreliability of information retrieval based bug localization tools. *Empir Software Eng* 22, 4 (2017), 2237–2279.
[113] T. O. A. Lehtinen, J. Itkonen, and C. Lassenius. 2017. Recurring opinions or productive improvements—what agile teams actually discuss in retrospectives. *Empir Software Eng* 22, 5 (2017), 2409–2452.
[114] P. Lenberg, R. Feldt, and L.G. Wallgren. 2015. Behavioral Software Engineering: a Definition and Systematic Literature Review. *J Syst Softw* 107 (2015), 15–37.
[115] P. Lenberg, L. G. W. Tengberg, and R. Feldt. 2017. An initial analysis of software engineers' attitudes towards organizational change. *Empir Software Eng* 22, 4 (2017), 2179–2205.
[116] F.J. Lerch, D.J. Ballou, and D.E. Harter. 1997. Using simulation-based experiments for software requirements engineering. *Annals of Software Engineering* 3, 1 (1997), 345–366.
[117] T.C. Lethbridge, S.E. Sim, and J. Singer. 2005. Studying Software Engineers: Data Collection Techniques for Software Field Studies. *Empir. Software Eng.* 10 (2005), 311–341.
[118] H. Li, W. Shang, and A. E. Hassan. 2017. Which log level should developers choose for a new logging statement? *Empir Software Eng* 22, 4 (2017), 1684–1716.
[119] H. Li, W. Shang, Y. Zou, and A. E. Hassan. 2017. Towards just-in-time suggestions for log changes. *Empir Software Eng* 22, 4 (2017), 1831–1865.
[120] D. Lin, C.-P. Bezemer, and A. E. Hassan. 2017. Studying the urgent updates of popular games on the Steam platform. *Empir Software Eng* 22, 4 (2017), 2095–2126.
[121] H.A. Linstone and M. Turoff (Eds.). 2002. *The Delphi Method Techniques and Applications.* Addison-Wesley.
[122] C. Lokan and E. Mendes. 2017. Investigating the use of moving windows to improve software effort prediction: a replicated study. *Empir Software Eng* 22, 2 (2017), 716–767.
[123] L. Lopez-Fernandez, G. Robles, and J.M. Gonzalez-Barahona. 2004. Applying social network analysis to the information in CVS repositories. In *Proc. 1st Workshop on Mining Software Repositories (MSR).* 101–105.
[124] Q. Luo, A. Nair, M. Grechanik, and D. Poshyvanyk. 2017. FOREPOST: finding performance problems automatically with feedback-directed learning software testing. *Empir Software Eng* 22, 1 (2017), 6–56.
[125] J. Ma, J. Li, W. Chen, R. Conradi, J. Ji, and C. Liu. 2008. A State-of-the-Practice Study on Communication and Coordination between Chinese Software Suppliers and Their Global Outsourcers. *Software Process Improvement and Practice* 13, 3 (2008), 233–247.
[126] L. MacLeod, A. Bergen, and M.-A. Storey. 2017. Documenting and sharing software knowledge using screencasts. *Empir Software Eng* 22, 3 (2017), 1478–1507.
[127] A. Mahmoud and G. Bradshaw. 2017. Semantic topic models for source code analysis. *Empir Software Eng* 22, 4 (2017), 1965–2000.
[128] R. Malhotra and M. Khanna. 2017. An empirical study for software change prediction using imbalanced data. *Empir Software Eng* 22, 6 (2017), 2806–2851.
[129] M. Martinez, T. Durieux, R. Sommerard, J. Xuan, and M. Monperrus. 2017. Automatic repair of real bugs in java: a large-scale experiment on the defects4j dataset. *Empir Software Eng* 22, 4 (2017), 1936–1964.
[130] J.E. McGrath. 1964. *Social Psychology: A Brief Introduction.* Holt, Rinehart and Winston, Inc.

[131] J.E. McGrath. 1964. Towards a "Theory of Method" for research on organizations. In *New Perspectives in Organization Research*, W. Cooper, H. Leavitt, and M. Shelly (Eds.). 533–556.
[132] J.E. McGrath. 1981. Dilemmatics: The Study of Research Choices and Dilemmas. *Am. Behav. Sci.* 25, 2 (1981), 179–210.
[133] J.E. McGrath. 1984. *Groups: Interaction and Performance.* Prentice-Hall, Inc.
[134] J.E. McGrath. 1994. Methodology Matters: Doing Research in the Behavioral and Social Sciences. In *Readings in Human-Computer Interaction: Toward the Year 2000*, Ronald M. Baecker (Ed.). 152–169.
[135] E.R. McLean. 1973. Comments on Empirical studies of management information systems by Richard L. Van Horn. *Data Base* 5, 2 (1973), 181–182.
[136] N. Medvidović and R. N. Taylor. 2000. A classification and comparison framework for software architecture description languages. *IEEE Trans Softw Eng* 26, 1 (2000), 70–93.
[137] T. Menzies, W. Nichols, F. Shull, and L. Layman. 2017. Are delayed issues harder to resolve? Revisiting cost-to-fix of defects throughout the lifecycle. *Empir Software Eng* 22, 4 (2017), 1903–1935.
[138] T. Menzies, Y. Yang, G. Mathew, B. Boehm, and J. Hihn. 2017. Negative results for software effort estimation. *Empir Software Eng* 22, 5 (2017), 2658–2683.
[139] B. Meyer, H. Gall, M. Harman, and G. Succi. 2013. Empirical Answers to Fundamental Software Engineering Problems (Panel). In *Proceedings of ESEC/FSE'13*. 14–18.
[140] M.W. Mkaouer, M. Kessentini, M. Ó Cinnéide, S. Hayashi, and K. Deb. 2017. A robust multi-objective approach to balance severity and importance of refactoring opportunities. *Empir Software Eng* 22, 2 (2017), 894–927.
[141] A. Mockus, R.T. Fielding, and J.D. Herbsleb. 2000. A case study of open source software development: the Apache server. In *Proc. International Conf. Software Engineering*.
[142] A. Mockus, R.T. Fielding, and J.D. Herbsleb. 2002. Two case studies of open source software development: Apache and Mozilla. *ACM Trans. Softw. Eng. Methodol.* 11, 3 (2002).
[143] M. Montesi and P. Lago. 2008. Software engineering article types: An analysis of the literature. *J Syst Softw* 81, 10 (2008), 1694–1714.
[144] M. Müller and D. Pfahl. 2008. Simulation Methods. In *Guide to Advanced Software Engineering*, F. Shull, J. Singer, and D. I.K. Sjøberg (Eds.).
[145] N. Munaiah, S. Kroh, C. Cabrey, and M. Nagappan. 2017. Curating GitHub for engineered software projects. *Empir Software Eng* 22, 6 (2017), 3219–3253.
[146] C.J. Neill and P. Laplante. 2003. Requirements Engineering: The State of the Practice. *IEEE Softw* 20, 6 (2003).
[147] L. Nguyen and G. Shanks. 2008. A framework for understanding creativity in requirements engineering. *Inform Software Tech* 51, 3 (2008), 655–662.
[148] A. Niknafs and D. Berry. 2017. The impact of domain knowledge on the effectiveness of requirements engineering activities. *Empir Software Eng* 22, 1 (2017), 80–133.
[149] H. Niu, I. Keivanloo, and Y. Zou. 2017. Learning to rank code examples for code search engines. *Empir Software Eng* 22, 1 (2017), 259–291.
[150] E. Noei, M. D. Syer, Y. Zou, A. E. Hassan, and I. Keivanloo. 2017. A study of the relation of mobile device attributes with the user-perceived quality of Android apps. *Empir Software Eng* 22, 6 (2017), 3088–3116.
[151] P. Ovaska, M. Rossi, and P. Marttiin. 2003. Architecture as a Coordination Tool in Multi-site Software Development. *Softw. Process Improve. Pract.* 8 (2003), 233–247.
[152] C. Palomares, C. Quer, and X. Franch. 2017. Requirements reuse and requirement patterns: a state of the practice survey. *Empir Software Eng* 22, 6 (2017), 2719–2762.
[153] J. Park, M. Kim, and D.-H. Bae. 2017. An empirical study of supplementary patches in open source projects. *Empir Software Eng* 22, 1 (2017), 436–473.
[154] D.L. Parnas. 2009. Point/Counterpoint: Empirical Research in Software Engineering: A Critical View. *IEEE Softw* 26, 6 (2009), 56–59.
[155] D.E. Perry, A.E. Porter, and L.G. Votta. 2000. Empirical Studies of Software Engineering: A Roadmap. In *Future of Software Engineering* (Limerick, Ireland).
[156] D.E. Perry, N.A. Staudenmayer, and L.G. Votta. 1994. People, Organizations, and Process Improvement. *IEEE Softw* 11, 4 (1994).
[157] K. Petersen and C. Wohlin. 2009. Context in Industrial Software Engineering Research. In *Proc. 3rd International Symposium on Empirical Software Engineering and Measurement.*
[158] S.L. Pfleeger and B.A. Kitchenham. 2001. Principles of Survey Research: Part 1: Turning Lemons into Lemonade. *ACM SIGSOFT Software Engineering Notes* 26, 6 (2001), 16–18.
[159] A.A. Porter, L.G. Votta, and V.R. Basili. 1995. Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment. *IEEE Trans Softw Eng* 21, 6 (1995).
[160] D. Raffo and S.-O. Setamanit. 2005. A Simulation model for global software development project. In *Proc. 6th International Workshop on Software Process Simulation and Modeling (ProSimâĂŹ05)* (Stuttgart, Germany). Fraunhofer IRB Verlag.

[161] P. Ralph. 2018. Toward Methodological Guidelines for Process Theories and Taxonomies in Software Engineering. *IEEE Transactions on Software Engineering* in press (2018).
[162] B. Ray, D. Posnett, V. Filkov, and P. Devanbu. 2014. A large scale study of programming languages and code quality in github. In *Proc. 22nd ACM SIGSOFT International Sym. Foundations of Software Engineering*.
[163] M. Riaz, J. King, J. Slankas, L. Williams, F. Massacci, C. Quesada-López, and M. Jenkins. 2017. Identifying the implied: Findings from three differentiated replications on the use of security requirements templates. *Empir Software Eng* 22, 4 (2017), 2127–2178.
[164] H. Robinson, J. Segal, and H. Sharp. 2007. Ethnographically-informed empirical studies of software practice. *Inform Software Tech* 49, 6 (2007), 540–551.
[165] J. Miguel Rojas, M. Vivanti, A. Arcuri, and G. Fraser. 2017. A detailed investigation of the effectiveness of whole test suite generation. *Empir Software Eng* 22, 2 (2017), 852–893.
[166] D. Rosenblum and E. Weyuker. 1996. Lessons learned from a regression testing case study. In *Proc. International Workshop on Empirical Studies of Software Maintenance (WESS'96)*.
[167] D.T. Ross. 1977. Guest Editorial: Reflections on Requirements. *IEEE Trans Softw Eng* 3, 1 (1977), 2–5.
[168] P. Runeson and M. Höst. 2009. Guidelines for conducting and reporting case study research in software engineering. *Empir Software Eng* 14 (2009), 131–164.
[169] P. Runeson, M. Höst, A. Rainer, and B. Regnell. 2012. *Case Study Research in Software Engineering: Guidelines and Examples*. Wiley.
[170] P.J. Runkel and J.E. McGrath. 1972. *Research on Human Behavior: A Systematic Guide to Method*. Holt, Rinehart and Winston, Inc.
[171] A. Sabané, Y.-G. Guéhéneuc, V. Arnaoudova, and G. Antoniol. 2017. Fragile base-class problem, problem? *Empir Software Eng* 22, 5 (2017), 2612–2657.
[172] V. Sakhnini, L. Mich, and D.M. Berry. 2017. Group versus individual use of power-only EPMcreate as a creativity enhancement technique for requirements elicitation. *Empir Software Eng* 22, 4 (2017), 2001–2049.
[173] A. Ashok Sawant and A. Bacchelli. 2017. fine-GRAPE: fine-grained APi usage extractor – an approach and dataset to investigate API usage. *Empir Software Eng* 22, 3 (2017), 1348–1371.
[174] C.B. Seaman. 1999. Qualitative Methods in Empirical Studies of Software Engineering. *IEEE Trans Softw Eng* 24, 4 (1999), 557–572.
[175] J. Segal. 2003. Some parallels between empirical software engineering and research in human-computer interaction. In *Proc. 15th Workshop of the Psychology of Programming Interest Group* (Keele, UK).
[176] S.-O. Setamanit. 2007. *A Software Process Simulation Model of Global Software Development (GSD) Projects*. Ph.D. Dissertation. Portland State University.
[177] S.-O. Setamanit, W. Wakeland, and D. Raffo. 2007. Using Simulation to Evaluate Global Software Development Task Allocation Strategies. *Softw. Process Improve. Pract.* 12 (2007), 491–503.
[178] B. Sharif, J. Meinken, T. Shaffer, and H. Kagdi. 2017. Eye movements in software traceability link recovery. *Empir Software Eng* 22, 3 (2017), 1063–1102.
[179] H. Sharp, Y. Dittrich, and C.R.B. de Souza. 2016. The Role of Ethnographic Studies in Empirical Software Engineering. *IEEE Trans Softw Eng* 42, 8 (2016), 786–804.
[180] H. Sharp and H. Robinson. 2004. An Ethnographic Study of XP Practice. *Empir Software Eng* 9, 4 (2004), 353–375.
[181] H. Sharp, M. Woodman, and F. Hovenden. 2005. Using Metaphor to Analyse Qualitative Data: Vulcans and Humans in Software Development. *Empir Software Eng* 10, 3 (2005), 343–365.
[182] M. Shaw. 2002. What Makes Good Research in Software Engineering? *Int J Softw Tools Technol Transf.* 4, 1 (2002).
[183] M. Shaw. 2003. Writing Good Software Engineering Research Papers. In *Proc. 25th International Conf. Software Engineering*. 726–736.
[184] Y. Shi, M. Li, S. Arndt, and C. Smidts. 2017. Metric-based software reliability prediction approach and its application. *Empir Software Eng* 22, 4 (2017), 1579–1633.
[185] F. Shull, J. Singer, and D.I.K. Sjøberg (Eds.). 2008. *Guide to Advanced Empirical Software Engineering*. Springer.
[186] J. Siegmund, N. Siegmund, and S. Apel. 2015. Views on Internal and External Validity in Empirical Software Engineering. In *Proc. 37th International Conference on Software Engineering*. IEEE. 10.1109/ICSE.2015.24.
[187] S.E. Sim, J. Singer, and M.-A. Storey. 2001. Beg, Borrow, or Steal: Using Multidisciplinary Approaches in Empirical Software Engineering Research. *Empir Softw Eng* 6, 1 (2001), 85–93.
[188] D.I.K. Sjøberg, B. Anda, E. Arisholm, T. Dybå, M. Jørgensen, A. Karahasanovic, E.F. Koren, and M. Vokác. 2002. Conducting Realistic Experiments in Software Engineering. In *Proc. International Symposium on Empirical Software Engineering (ISESE'ĂŹ02)*.
[189] D.I.K. Sjøberg, T. Dybå, B.C.D. Anda, and J.E. Hannay. 2008. Building Theories in Software Engineering. In *Guide to Advanced Empirical Software Engineering*, Forrest Shull, Janice Singer, and Dag I.K. Sjøberg (Eds.).
[190] D.I.K. Sjøberg, T. Dybå, and M. Jørgensen. 2007. The Future of Empirical Methods in Software Engineering Research. In *Future of Software Engineering*. IEEE Computer Society.
[191] D.I.K. Sjøberg, J.E. Hannay, O. Hansen, V.B. Kampenes, A. Karahansanovic, N.-K. Liborg, and A.C. Rekdal. 2005. A Survey of Controlled Experiments in Software Engineering. *IEEE Trans Softw Eng* 31, 9 (2005).

[192] A.E. Kelley Sobel and M.R. Clarkson. 2002. Formal Methods Application: An Empirical Tale of Software Development. *IEEE Trans Softw Eng* 28, 3 (2002), 308–320.
[193] A.E. Kelley Sobel and M.R. Clarkson. 2003. Response to "Comments on 'Formal Methods Application: An Empirical Tale of Software Development"'. *IEEE Trans Softw Eng* 29, 6 (2003).
[194] D. Spinellis. 2017. A repository of Unix history and evolution. *Empir Software Eng* 22, 3 (2017), 1372–1404.
[195] D. Ståhl, K. Hallén, and J. Bosch. 2017. Achieving traceability in large scale continuous integration and delivery deployment, usage and validation of the eiffel framework. *Empir Software Eng* 22, 3 (2017), 967–995.
[196] I. Stavropoulou, M. Grigoriou, and K. Kontogiannis. 2017. Case study on which relations to use for clustering-based software architecture recovery. *Empir Software Eng* 22, 4 (2017), 1717–1762.
[197] K. Stol, B. Caglayan, and B. Fitzgerald. 2018. Competition-Based Crowdsourcing Software Development: A Multi-Method Study from a Customer Perspective. *IEEE Trans Softw Eng* in press (2018).
[198] K. Stol and B. Fitzgerald. 2014. Two's Company, Three's a Crowd: A Case Study of Crowdsourcing Software Development. In *Proc. 36th International Conference on Software Engineering* (Hyderabad, India). 187–198.
[199] K. Stol and B. Fitzgerald. 2015. A Holistic Overview of Software Engineering Research Strategies. In *Third International Workshop on Conducting Empirical Studies in Industry (CESI)* (Florence, Italy). ACM.
[200] K. Stol and B. Fitzgerald. 2015. Theory-Oriented Software Engineering. *Science of Computer Programming* 101 (2015), 79–98.
[201] K. Stol, M. Goedicke, and I. Jacobson. 2016. Introduction to the special section—General Theories of Software Engineering: New advances and implications for research. *Inform Software Tech* 70 (2016), 176–180.
[202] K. Stol, P. Ralph, and B. Fitzgerald. 2016. Grounded Theory in Software Engineering Research: A Critical Review and Guidelines. In *Proc. 38th International Conference on Software Engineering* (Austin, TX, USA). ACM, 120–131.
[203] M.-A. Storey, L. Singer, F.F. Filho, A. Zagalsky, and D.M. German. 2017. How social and communication channels shape and challenge a participatory culture in software development. *IEEE Trans Softw Eng* 43, 2 (2017), 185–204. 10.1109/TSE.2016.2584053.
[204] P. Thongtanunam, S. McIntosh, A. E. Hassan, and H. Iida. 2017. Review participation in modern code review. *Empir Software Eng* 22, 2 (2017), 768–817.
[205] N.M. Tichy, M.L. Tushman, and C. Fombrun. 1979. Social Network Analysis For Organizations. *Acad. Manag. Rev.* 4, 4 (1979), 507–519.
[206] W.F. Tichy. 1998. Should Computer Scientists Experiment More? *Computer* 31, 5 (1998), 32–40.
[207] W.F. Tichy. 2000. Hints for Reviewing Empirical Work in Software Engineering. *Empir Softw Eng* 5 (2000), 309–312.
[208] J. Tisseau. 2001. Virtual Reality : in virtuo autonomy. (2001). University of Rennes I.
[209] A. Tosun, O. Dieste, D. Fucci, S. Vegas, B. Turhan, H. Erdogmus, A. Santos, M. Oivo, K. Toro, J. Jarvinen, and N. Juristo. 2017. An industry experiment on the effects of test-driven development on external quality and productivity. *Empir Software Eng* 22, 6 (2017), 2763–2805.
[210] G.H. Travassos and M. de Oliveira Barros. 2003. Contributions of In Virtuo and In Silico Experiments for the Future of Empirical Studies in Software Engineering. In *Proc. 2nd Workshop on Empirical Software Engineering*.
[211] C. Tsigkanos, L. Pasquale, C. Menghi, C. Ghezzi, and B. Nuseibeh. 2014. Engineering Topology Aware Adaptive Security: Preventing Requirements Violations at Runtime. In *Proc. 22nd IEEE International Requirements Engineering Conference* (Karlskrona, Sweden). 203–212.
[212] R.L. van Horn. 1973. Empirical studies of management information systems. *ACM SIGMIS Database: the DATABASE for Advances in Information Systems* 5 (1973), 172–182.
[213] J. van Maanen. 1982. Fieldwork on the Beat. In *Varieties of Qualitative Research*, J. van Maanen, J.M. Dabbs, and R.R. Faulkner (Eds.). Sage Publications.
[214] C. Vendome, G. Bavota, M. Di Penta, M. Linares-Vásquez, D. German, and D. Poshyvanyk. 2017. License usage and changes: a large-scale study on gitHub. *Empir Software Eng* 22, 3 (2017), 1537–1577.
[215] P. Vitharana. 2017. Defect propagation at the project-level: results and a post-hoc analysis on inspection efficiency. *Empir Software Eng* 22, 1 (2017), 57–79.
[216] L. Votta. 1995. By the Way, Has Anyone Studied Any Real Programmers, Yet?. In *Proc. 9th International Software Process Workshop*.
[217] E.J. Webb, D.T. Campbell, R.D. Schwartz, and L. Sechrest. 1966. *Unobtrusive measures: Nonreactive research in the social sciences*. Rand-McNally.
[218] G.M. Weinberg. 1971. *The psychology of computer programming*. Van Nostrand Reinhold New York.
[219] R. Wieringa. 2009. Design Science as Nested Problem Solving. In *Proc. DESRIST*.
[220] R. Wieringa and M.G. Heerkens. 2006. The methodological soundness of requirements engineering papers: a conceptual framework and two case studies. *Requir Eng* 11 (2006), 295–307. 10.1007/s00766-006-0037-6.
[221] R. Wieringa, N. Maiden, N. Mead, and C. Rolland. 2006. Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. *Requir Eng* 11 (2006), 102–107. 10.1007/s00766-005-0021-6.
[222] C. Wohlin and A. Aurum. 2015. Towards a decision-making structure for selecting a research design in empirical software engineering. *Empir Software Eng* 20, 6 (2015).

[223] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, and A. Wesslén. 2000. *Experimentation in Software Engineering*. Kluwer Academic Publishers.
[224] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, and A. Wesslén. 2012. *Experimentation in Software Engineering* (2nd ed.). Springer.
[225] C. Wohlin, D. Šmite, and N.B. Moe. 2015. A general theory of software engineering: Balancing human, social and organizational capitals. *J Sys Softw* 109 (2015), 229–242.
[226] J. Wu, S. Ali, T. Yue, J. Tian, and C. Liu. 2017. Assessing the quality of industrial avionics software: an extensive empirical evaluation. *Empir Software Eng* 22, 4 (2017), 1634–1683.
[227] X. Xia, L. Bao, D. Lo, P.S. Kochhar, A. E. Hassan, and Z. Xing. 2017. What do developers search for on the web? *Empir Software Eng* 22, 6 (2017), 3149–3185.
[228] D. Ye, Z. Xing, and N. Kapre. 2017. The structure and dynamics of knowledge network in domain-specific Q&A sites: a case study of stack overflow. *Empir Software Eng* 22, 1 (2017), 375–406.
[229] R.K. Yin. 2003. *Case Study Research: Design and Methods*. Sage.
[230] C. Zannier, G. Melnik, and F. Maurer. 2006. On the Success of Empirical Studies in the International Conference on Software Engineering. In *Proc. International Conf. Software Engineering*. 341–350.
[231] M.V. Zelkowitz. 2007. Techniques for Empirical Validation. *Empirical Software Engineering Issues* LNCS 4336 (2007), 4–9.
[232] M.V. Zelkowitz and D.R. Wallace. 1998. Experimental Models for Validating Technology. *Computer* 31, 5 (1998), 23–31.
[233] W. Zogaan, I. Mujhid, J. C. S. Santos, D. Gonzalez, and M. Mirakhorli. 2017. Automated training-set creation for software architecture traceability problem. *Empir Software Eng* 22, 3 (2017), 1028–1062.

Received November 2017; revised March 2018; revised June 2018; accepted July 2018