Empirical Software Engineering manuscript No.
(will be inserted by the editor)

Methodology Matters: How We Study Socio-Technical Aspects in Software Engineering

Margaret-Anne Storey · Courtney Williams · Neil A. Ernst · Alexey Zagalsky · Eirini Kalliamvakou

Received: date / Accepted: date

Abstract Modern software engineering involves both human and technical aspects, the importance of which is widely accepted by practitioners and researchers alike. At a community level, software engineering researchers may be expected to choose a balance of research strategies that capture both social and technical characteristics of software development. In this paper, we consider if the research strategies we use do in fact provide this balance. We first developed a research strategy framework to help distinguish research strategies that directly study human and social aspects, from strategies that rely on data such as trace, archival or simulated data, and those that may focus more on technical or system aspects. We utilized this framework to categorize the research strategies used by 253 technical track papers from the International Conference on Software Engineering (ICSE). Using a design science lens, we further identified the types of research contributions provided in these papers—either descriptive knowledge, or the design and evaluation of technical solutions. We mapped the contribution types to the research strategies identified. We found that, at the community level, the papers we analyzed strongly favour data strategies over strategies that directly study human and social aspects, and most research contributions consist of the design or evaluation of technical solutions. We conclude by proposing that our community should diversify our use of research strategies so that we may have a deeper understanding of human and social aspects of software development practice, while balancing the design and evaluation of innovations on the technical side.

Keywords Empirical methods · Human studies · Software engineering, Meta-research · Survey

Margaret-Anne Storey · Courtney Williams · Neil A. Ernst · Eirini Kalliamvakou
University of Victoria, Canada
E-mail: mstorey@uvic.ca, courtneyelwilliams@gmail.com, nernst@uvic.ca, ikaliam@uvic.ca

Alexey Zagalsky
Coller School of Management, Tel Aviv University, Israel
E-mail: alexeyza@gmail.com

1 Introduction

Software engineering is at the forefront of innovation and research, and involves the consideration of both human and technical aspects. The origins of software engineering come from the 1950s and 1960s when the field emerged as a sub-discipline of computer science and engineering. As such, it was highly technical and focused on solving technical, logical and mathematical problems and solutions. However, seminal works have gradually drawn attention to the importance of developers and social factors in software engineering [4,7,26,34], as well as the need to understand software development processes. Nowadays, software development is widely recognized as a socio-technical endeavor [35]: many researchers consider both technical and human aspects of software development in their efforts to understand software engineering practices and improve tools, and special-interest communities with a focus on human aspects (such as the CHASE community[1]) have been formed.

To study human aspects of theories and tools, researchers can take advantage of several specific research methodologies that make use of empirical strategies suitable for studying these aspects. The qualitative methods described by Seaman *et al.* [22] are particularly valuable in highlighting human aspects in software engineering research. Sharp *et al.* [23] advocate using **ethnography** in software engineering studies, pointing to its ability to capture what developers do in practice, *why* they follow certain processes, or *how* they use certain tools. Kitchenham [12] calls for incorporating approaches from social science, such as **case studies** and **quasi-experiments**, as she argues it will make findings about theories or tools more relevant to practitioners. Sjøberg, Dybå, and Jørgensen [30] argue that doing more empirical work in software engineering (SE) will provide us with the knowledge needed to develop better technologies and processes for software development. And although empirical research in software engineering has increased, in this paper we question how many of our studies directly study developers or other stakeholders and do not focus solely on technical aspects.

To understand how human aspects are studied (or not), we conducted a meta-study examining the *research strategies*—a category of more detailed research methods—reported in a cohort of papers published at the International Conference on Software Engineering (ICSE). We chose ICSE because it is seen by many as the flagship conference in software engineering, and is the venue for "the most recent innovations, trends, experiences and concerns in the field of software engineering."[2] Many of the papers presented at ICSE propose or evaluate technical tools and/or present descriptive knowledge as their main contribution. We also considered and mapped which research strategies were used for generating *descriptive knowledge* and which ones were used for designing or evaluating specific *tools and/or solutions.* It is reasonable to expect that certain ICSE papers would not directly involve developers in their stud-

[1] http://www.chaseresearch.org/home
[2] http://www.icse-conferences.org

ies, and may indirectly study developer behaviour through simulation or by mining and analyzing developer trace or archival data. However, at a community level, we would expect to find some proportion of papers that study developer behaviours directly and evaluate processes, tools and interventions in real-world developer work contexts.

To study how human aspects are investigated in software engineering research at ICSE, we considered the following two research questions:

RQ1: What *research strategies* are described in research published at ICSE?
RQ2: What type of *research contributions* do we find in research published at ICSE, and how do the research strategies we find map to these contribution types?

We categorized both the strategies and contributions of 253 papers published in the technical track at ICSE (accepted papers over a recent 3-year period, 2015 to 2017). As mentioned above, we chose to study ICSE because it is highly regarded and is assumed to cover a broad set of sub-topics or research strategies. To answer our first research question, we developed a framework that helps distinguish research strategies that directly measure human and social aspects (e.g., an observational field study or a survey) from strategies that do not (e.g., a data mining study or simulation study). To answer our second research question, we categorized the papers according to the type of research contribution. For this we used an existing *design science lens* [9] which helped us identify if the research contribution constituted descriptive knowledge (discovery of what is), or if the main contribution constitued an intervention such as tool or process innovation.

In the papers we considered, we found they were skewed towards research strategies that do not directly study human and social aspects: these are research strategies that rely on the use of trace, archival or simulated data. We also found that research contributions in these papers were more focused on technical solution contributions over descriptive knowledge contributions.

The remainder of this paper is structured as follows. In Section 2, we discuss related work that has both informed and motivated this research, and we discuss other research that has reflected on the types of research we engage in. In Section 3, we then describe our research methodology and present our research strategy framework and review the design science lens we used in our research. In Section 4, we present the findings of our study. We discuss possible explanations for the results that are grounded in our data in Section 5 and offer recommendations for our research community and individual researchers to consider. We discuss limitations of this research in Section 6. Finally, we conclude by identifying areas for future work and reiterating important takeaways in Section 7. Traceability artifacts from our analysis and a replication package are published on our supplementary website at https://bit.ly/2ZkF4au.

2 Background

Software development is a highly complex and technical process, and developers use many different technologies to design, develop, deploy, and maintain software. While much of our research is technical, the importance of the human and social factors of software development has been recognized since the early days of software engineering. Books such as *The Psychology of Computer Programming* [34] and *Software Psychology: Human Factors in Computer and Information Systems* [26] called attention to the important role of developers and social factors in software development. Other authors drew on personal experiences to demonstrate the impact of different social constructs and management practices in software development, including *The Mythical Man-Month* [4] and *Peopleware: Productive Projects and Teams* [7]. While these are only a few of the many examples of early social research in software engineering, they remain relevant today.

To study complex socio-technical systems, software engineering researchers must employ a wide variety of techniques from a diverse set of interdisciplinary fields. There are a number of seminal works that provide guidance for conducting and reflecting on empirical research in software engineering. One key example is the book *Empirical Methods and Studies in Software Engineering* [6], published in 2003. It offers an introduction of four major empirical methods: "controlled experiments, case studies, surveys, and postmortem analyses" [36]. Another prominent research book, published in 2007, is the *Guide to Advanced Empirical Software Engineering* [27]. This book includes guidance for a number of specific techniques, including qualitative methods [22], focus groups [15], personal opinion surveys [13], and data collection techniques for field studies [28]. It also provides guidance on general topics, such as how to design ethical studies involving humans in software engineering [33], a guide for building theories in software engineering [29], and a chapter explaining the benefits and drawbacks of different empirical methods in software engineering to assist in research design choices [8].

Other researchers offer guidance on specific methods for studying human subjects: Stol, Ralph, and Fitzgerald provide guidelines for grounded theory in the context of software engineering [51] as they found that many papers reporting the use of grounded theory lacked rigor. Runeson and Høst adapted case study research guidelines to the software engineering domain [20], also in part to address the misuse of the term case study in our community. There are also a number of seminal works available that focus around experimentation and evaluations. Both Wohlin *et. al.* [37] and Ko, Latoza, and Burnett [14] provide excellent resources for understanding how to conduct software engineering experiments with human participants. Sharp *et al.* [23] recently explained how ethnographic studies could show not only what developers do in practice but also *why*, and encouraged SE researchers to incorporate ethnography into their empirical studies.

These methods for directly studying human activities and behaviours are used across our community. Typically there is at least one track on human aspects in the main research conferences, as well as special purpose workshops on the topic, such as the CHASE series[3]. The papers presented at CHASE tend to address broad socio-technical topics, but the workshop focuses on early results. The Empirical Software Engineering and Measurement (ESEM) conference and Empirical Software Engineering (EMSE) journal also attract papers that consider human aspects, as their focus is on empirical methods, many of which directly involve humans. But how frequently human aspects are considered in our main venues, particularly in papers that present technical innovations, is not at all evident. And some researchers feel that the coverage of human and social aspects is lacking [17, 24].

There are several meta-studies that reflect on papers published in our community and our use of empirical methods. Zelkowitz [38] found that the community's use of empirical validation techniques for research contributions was improving, but that researchers were using terms such as "case study" to refer to different levels of abstraction, making it hard to understand the communicated research. More recently, Siegmund *et al.*'s work [50] prompted a discussions about research methodology choice and its impact on external and internal validity within our community. Another recent paper, published by Stol and Fitzgerald, builds on Runkel and McGrath's research framework to provide consistent terminology for research strategies [31]. They adapted the research framework to categorize research studies but not with a specific focus on human aspects.

In terms of categorizing papers by type of research contribution, Shaw [25] investigated the papers submitted to ICSE 2002, analyzing the content of the papers that were both accepted and rejected, as well as observing program committee conversations about which papers to accept. She found that there were very low rates of submission and acceptance of papers that investigated "categorization" or "exploration" research questions, or papers whose research results presented "qualitative or descriptive models". A 2016 replication of Shaw's methodology [32] found that a new category of research papers, mining software repositories, was common. This new category of mining papers may study human behaviours, but often in an indirect way through the trace data or archival data developers or users may leave behind.

The software engineering landscape is constantly changing with the creation of new technologies and it has shifted in recent years with the addition of platforms and tools that make software development more collaborative and social (GitHub, Stack Overflow, and Slack are prominent examples). Our aim is to understand how the software engineering research community studies *human aspects* in software engineering, and which kinds of contributions we arrive at. An investigation of how social aspects are captured and discussed in current software engineering research is therefore both relevant and timely.

[3] Cooperative and Human Aspects of Software Engineering, co-located with ICSE since 2011 `http://www.chaseresearch.org/`

3 Methodology

To answer our research questions (see Section 1), we manually analyzed and categorized the *research strategies* used (RQ1) and types of *research contributions* across three years of ICSE papers (RQ2). We considered all technical research track papers from ICSE's 2015, 2016, and 2017 proceedings in the sample, collecting 84, 101, and 68 technical track papers from each year, respectively, for a total of 253 papers. We focused on three years of ICSE because it is the flagship SE conference and not focused on any one specific subfield of SE research. We also wanted to understand the current state of ICSE rather than show trends over time.

For the purposes of replication and traceability, we provide our methodological tools, the anonymized raw data, and analysis documents on our supplementary website, `https://bit.ly/2ZkF4au`.

To address RQ1, we categorized our cohort of ICSE papers using a research strategy framework we developed that categorizes strategies and highlights which ones directly study human and social aspects. We describe the research strategy framework in Subsection 3.1. To address RQ2, we also categorized the main research contributions in each of the papers using a design science lens [9]. We provide an overview of this design science lens in Subsection 3.2.

3.1 A Research Strategy Framework: Interpreting Socio-Technical Research in Software Engineering

We developed a research strategy framework, shown in Fig. 1, as a way for interpreting and categorizing socio-technical research in software engineering. This framework is derived from Runkel and McGrath's model of research strategies [18, 21].

Runkel and McGrath's model of research strategies was developed in the 1970s for categorizing human behavioral research, hence it provides a good model for examining socio-technical factors in software engineering. The McGrath model has been used by other software engineering researchers to reflect on research strategy choice and its implications on research design [8]. It has also been used extensively in the field of Human Computer Interaction to guide research design. Our framework separates empirical from non-empirical research. Three of the four main empirical categories are lab, field and respondent strategies (in line with Runkel and McGrath's framework). We add a new category of empirical research strategies, data strategies, to our framework that refers to research that relies on data rather than direct participation or observation of human subjects. Data strategies may be used to study system aspects only or to study socio-technical aspects (as the data studied may reflect previous human activities).

Runkel and McGrath's model highlights how the choice of research strategy impacts desirable research criteria: generalizability, realism, and control. *Generalizability* refers to how generalizable the findings may be to the population

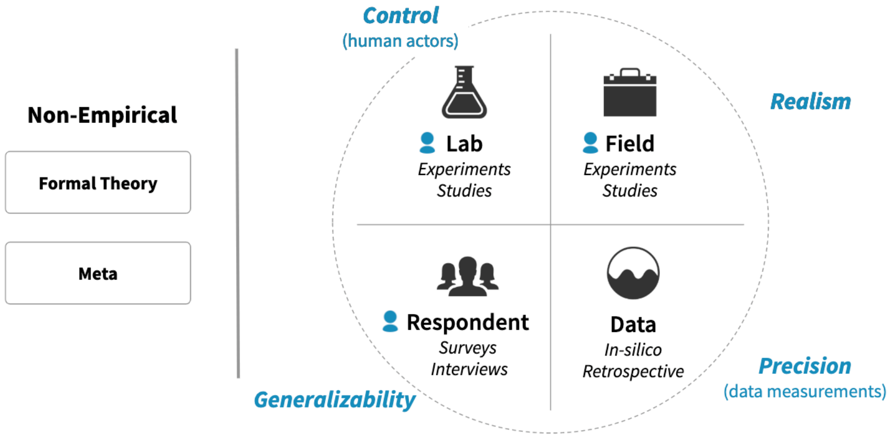

Fig. 1 Our research strategy framework for categorizing research strategies from a socio-technical perspective.

outside of the specific actors under study. *Realism* refers to how closely the context under which evidence is gathered matches real life. *Control* refers to the control over the measurement of variables that may be relevant when human behaviors are studied. We added a fourth criterion to our framework, data *precision*, as some methods have higher potential for collecting precise measurements of system data over other methods. Acknowledging that all strategy choices have inherent weaknesses, Runkel and McGrath emphasized the importance of *triangulation* across research strategies as a mitigation strategy. We describe the research strategies in our framework next.

Field strategies in software engineering involve researchers entering the natural development setting to study the research participants (often developers, engineers or managers), with the potential to increase realism. For example, a *field study* is one field strategy where a researcher may observe, using ethnography, how agile practices are adopted in a startup company (increasing realism). A *field experiment* is another field strategy that may also occur in the natural development setting but the researcher will control aspects of the setting which may be more obtrusive on the study participants. For example, a novel automatic testing tool may be introduced to observe its effects on code quality, but in the developers' realistic setting.

Lab strategies in software engineering involve testing hypotheses in highly controlled situations or environments but at the expense of realism. A *lab experiment* is one strategy that could involve a researcher investigating the effects of a new debugging tool (in comparison to the *status quo* debugger) on programming task efficiency with graduate students in a lab. By comparison, an *experimental simulation* may involve a researcher investigating project management meetings where they set up an experiment in a room with a similar setup to the one used at the company, potentially leading to increased realism at the expense of control over variables that come into play when human actors are involved.

Respondent strategies are commonly used in software engineering research because capturing self-reported data from human participants is more convenient than with field or lab studies where the experimenter collects the data. A *sample survey* is one respondent strategy which may, for example, involve an online survey or a set of interviews to learn how continuous integration tools are used by developers and which challenges they may encounter using these tools. A *judgment study*, another respondent strategy, may be used to evaluate the effectiveness of a new tool or process, by asking participants to try out the new tool in a setting of convenience and provide their opinion about the new tool over existing tools. For example, a researcher may wish to ask students for their opinions about a new debugging tool by asking them to use it in the classroom and provide their opinion over other tools they have used.

Data strategies do not directly study human behaviours and involve the use of either previously collected data—which may have been generated by humans, e.g., source code or operational trace data such as bugs, commits, error logs or archival records—or they may involve simulated data. [4] Data strategies are commonly used in software engineering (as we will show) and use a wide variety of methods, including experiments to evaluate software tools, data mining studies, computational analysis of big data, the creation and evaluation of prediction models, natural language processing techniques, and computer simulations. Data strategies show higher potential for precise data measurements, but very low control over variables that may arise from human actors (if relevant). For example, a researcher may wish to evaluate a new bug detection technique and may use version control history data in an open-source project to determine if their tool identified all the bugs that were fixed in subsequent versions of the project. In this case, the researcher can measure the data in a precise way but cannot see how the developers may use the new technique in their natural environment.

Non-empirical strategies focus on the use of *formal methods* or the development of a theory, or they may refer to studies of the software research endeavor itself (e.g., via systematic literature reviews or meta-analysis). We refer to these as *meta studies*.[5] The results from meta studies are typically

[4] For readers familiar with Runkel and McGrath's research strategy circumplex, the data strategies quadrant is not found in the circumplex and is a new addition. The use of such methods was not that common in the 70s, although their model included computer simulation which refers to studies where a closed world is used for a simulation study. A data strategy could refer to a computer simulation or other data driven methods.

[5] This category replaces the non-empirical strategy quadrant (i.e., theory and computer simulation) in Runkel and McGrath's circumflex.

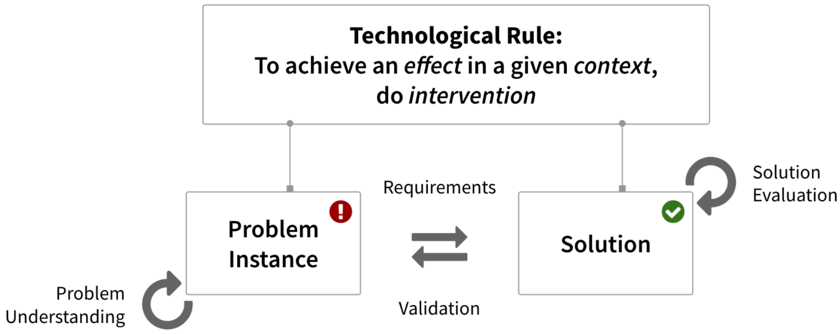

Fig. 2 The design science lens we used for distinguishing *descriptive* research from *solution*-oriented research in empirical papers (see Section 4.2). This lens also helped in distinguishing which *stakeholders* were targeted in the research study (see Section 5).

consumed and used by other researchers and are only indirectly applicable to practitioners.

3.2 A Design Science Lens: Categorizing Research Contributions

Design science is a paradigm for conducting and communicating applied research, such as software engineering. Similar to other design sciences, much software engineering research aims to understand the nature of practical problems and design solutions for those problems in a real-world context. The goal of design science research is to produce prescriptive knowledge for professionals in a discipline and to share empirical insights gained from investigations of the prescriptions applied in context [1]. Such knowledge is referred to as "design knowledge" as it helps practitioners design solutions to their problems.

In previous work, the first author of this paper (Storey), together with other colleagues, developed a design science visual abstract (see Fig. 2) to use as a lens for articulating the nature of research contributions [9].

To distinguish a paper's type of research contribution, the visual abstract can be used to distinguish descriptive knowledge from solution designs and evaluations. Descriptive knowledge, in terms of problem or context understanding, studies problem instances, as shown on the left side of Fig. 2, and may often be articulated as descriptive or predictive theories. Solution designs or knowledge about solutions, which may be framed as "technological rules", are theory fragments which capture how a particular intervention (tool innovation or process recommendation) can be applied in a given problem context and what effect it may achieve. Solution design studies may capture insights concerning the problem instance as well as the evaluation of the solution, either independently of a problem instance or by evaluating the solution directly on the problem instance (see Fig. 2). However, many papers that focus on solution designs tend to be focused on the right side of this figure only.

We used the design science lens in our earlier work to analyze and categorize research contributions from a cohort of papers published at ICSE (specifically, the ACM SIGSOFT Distinguished Paper Award papers from 2015-2018). We found that 27 of these papers involve the design and/or evaluation of a technical solution (and contributed a technological rule), while 8 papers contribute descriptive knowledge.

We use this lens in this paper to help us answer our second research question and to categorize which type of research contribution was found in each of the ICSE papers in the cohort we examined for this research. Specifically, we aimed to answer (for the papers that use an empirical research strategy) if the papers produce a **contribution of descriptive knowledge** and/or a **technical solution contribution** (in terms of its design and/or evaluation). We then mapped the research strategies identified for each empirical paper to the paper's type of contribution. We present the results to our research questions in the next section of this paper.

4 Findings

To understand how the SE research community investigates the social aspects of software development, we identified the research strategies and contributions in our cohort of ICSE papers. First, we consider the research strategies used (based on the research strategy framework we described in Fig. 3 above), then we consider the research contributions in these papers (descriptive and/or solution). Finally, we map the different research strategies to the types of research contributions we found.

4.1 RQ1: Which research strategies are used in the research published at ICSE?

Among the 253 papers we examined, we found a higher use of data strategies (195 papers reported the use of data strategies) compared to any of the other research strategies (see Fig. 3). Two examples of data strategy papers are first: Christakis *et al.*'s ICSE 2017 paper, "A General Framework for Dynamic Stub Injection" [39], which describes their novel stub injection tool, and their evaluation of the tool by running it on a series of real applications using previously collected data. The second example is Joblin *et al.*'s ICSE 2017 paper, "Classifying Developers into Core and Peripheral: An Empirical Study on Count and Network Metrics" [42], which analyzes commit data from GitHub projects to study aspects of human behavior and uses prediction algorithms to classify developers as core or peripheral in open source GitHub projects.

There were significantly fewer instances of the other empirical research strategies: Field (26), Lab (22) and Respondent (36). We discuss the possible

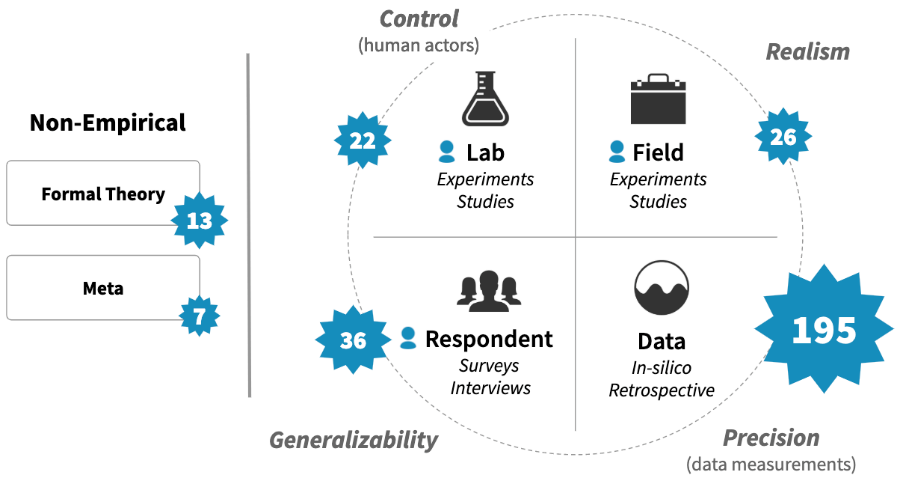

Fig. 3 Research strategies used in the 253 ICSE papers we analyzed. Some papers reported more that one method.

implications of this imbalance of research strategy use in Section 5, but first we give examples of each of the categories shown in Fig. 3.

Prechelt, Schmeisky and Zieris' ICSE 2016 paper, "Quality Experience: A Grounded Theory of Successful Agile Projects Without Dedicated Testers" [47], is one example of a field study from our set of papers. They conducted a series of on-site case studies of companies using Agile development methods to explore the impact of dedicated testers. Because the researchers did not intervene in the natural development environment, this allowed them to make realistic observations in an unobtrusive manner. In contrast, Sadowski, van Gogh and Jaspan introduced a new program analysis ecosystem to an existing development environment in their ICSE 2015 paper, "Tricorder: Building a Program Analysis Ecosystem" [48], which is an example of a field experiment. By introducing a new tool into the Google development environment, this somewhat obtruded on the human participants in the study, but at the benefit of having more control over variables that may unsettle human participants.

An example of a laboratory experiment in our sample is Vakilian *et al.*'s 2015 ICSE paper, "Cascade: A Universal Programmer-assisted Type Qualifier Inference Tool" [53]. They developed a novel interactive type qualifier inference tool and evaluated the tool's impact on code quality by conducting experiments using students in a lab. By creating a contrived development setting, they were able to exert a high amount of control, increasing the likelihood that the effects they observed in their experiment were caused by their manipulated variable (the use of the tool).

There were two common types of sample survey strategies in our sample: online questionnaires (surveys) and interviews. For example, Kononenko, Baysal, and Godfrey's paper from ICSE 2016, "Code Review Quality: How

Developers See It" [43], details their use of an online questionnaire to learn about developer perceptions around code review quality. Hoda and Noble's ICSE 2017 paper, "Becoming Agile: A Grounded Theory of Agile Transitions in Practice" [41], discussed the use of interviews to gather responses from various developers about their experiences transitioning to Agile development methods in their work. Because these researchers gathered responses from a wide variety of developers working in different settings, this increased the generalizability of their findings to other development contexts beyond those represented in the studies. Treude and Robillard's ICSE 2016 paper, "Augmenting API Documentation with Insights from Stack Overflow" [52], is a good example of a judgment study from our sample. They created a novel API recommendation system and invited developers to use the system in their work, and then inquired about the accuracy of the resulting recommended APIs. Having developers use the tool in their own work increased the generalizability of the findings because the tool was applied to more contexts, but also decreased control in the study as there could be other factors beyond the scope of the study influencing participant responses.

For the non-empirical strategies, 13 papers discussed formal methods or developed a formal theory, such as Faitelson and Tyszberowicz's ICSE 2017 paper, "UML Diagram Refinement (Focusing on Class- and Use Case Diagrams)" [40]. A total of 7 non-empirical strategy papers were aimed at other researchers (e.g., discussion of research methods), such as Stol, Ralph, and Fitzgerald's ICSE 2016 paper, "Grounded Theory in Software Engineering Research: A Critical Review and Guidelines" [51].

The data strategy papers reported a variety of research methods, including data mining studies, natural language processing experiments, computer simulations, computational experiments to evaluate tools and techniques, computational analysis of software artifacts, and computational prediction models. Most of these papers relied on the analysis of retrospective data, such as source code, log files, data from social websites like Stack Overflow, datasets of software bugs, open source repositories, and code comments.

Triangulation across research strategies is key for balancing the desirable research quality criteria of generalizability, control over variables that may be influenced by human actors, realism, and precision over data measurements. Some of the papers we analyzed reported up to three research strategies. For all papers, only 46 papers (18%) reported more than one research strategy. For the 195 data strategy papers, 37 papers triangulated the data research methods with other research strategies (see Fig. 4), while the remaining 158 papers did not report any triangulation. Linking the ICSE papers with papers in other venues that may triangulate findings was out of scope for our study. We examine the 158 data strategy papers that do not triangulate with other research strategies when we consider types of research contributions as part of RQ2.

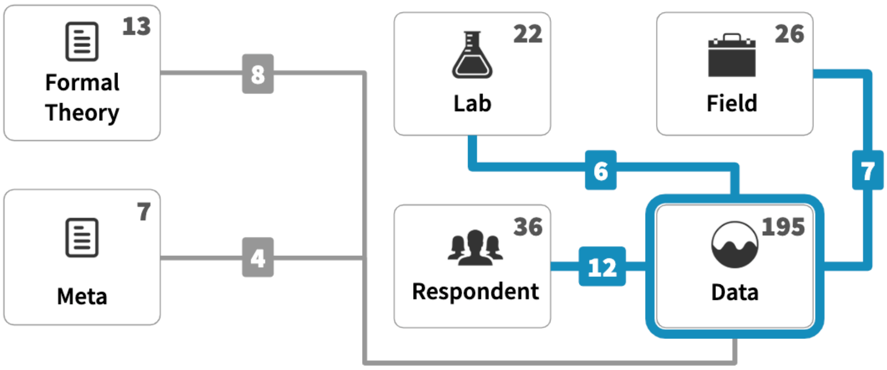

Fig. 4 Triangulation of data research strategies with all other strategies, as reported in our set of ICSE papers.

4.2 RQ2: What research contribution types are found in research published at ICSE and how do these map to research strategies?

In Section 3.2, we explained how we leveraged a design science lens to understand the contributions ICSE papers might make and how a paper's contributions could be framed in terms of relevance to stakeholders. We read each of the 253 papers and identified their primary *type of research contribution*. For example, in the 2017 ICSE paper titled "Software Development Waste", authors Todd Sedano, Paul Ralph and Cécile Péraire [49] conducted a study in the field using interviews and observations to derive a new theory. We coded this as having a primarily descriptive contribution (i.e., problem understanding according to the visual abstract of Fig. 2).

Looking at all 253 papers, we found that 66 papers (26%) were predominantly descriptive, and 187 (74%) were primarily solution oriented. These numbers line up closely with our results in the previous section (where 77% of papers used data strategies). This is not a coincidence: many of the data strategies were used in papers that designed a new solution. For example, the 2015 ICSE paper by Adrian Nistor, Po-Chun Chang, Cosmin Radoi and Shan Lu [46], "CARAMEL: Detecting and Fixing Performance Problems That Have Non-Intrusive Fixes", describes a solution to the problem of fixing performance bugs. It uses a data strategy to propose a new solution to the problem, but it does not examine the problem itself in detail as that is well described in previous work.

To elaborate on this mapping from strategy to research contribution type, we created the alluvial flow diagram shown in Fig. 5. It outlines the mappings between research strategy (RQ1) and research contribution type (RQ2). Notice the majority of non-data studies make descriptive research contributions, while the majority of data strategy studies concern technical solution contributions.

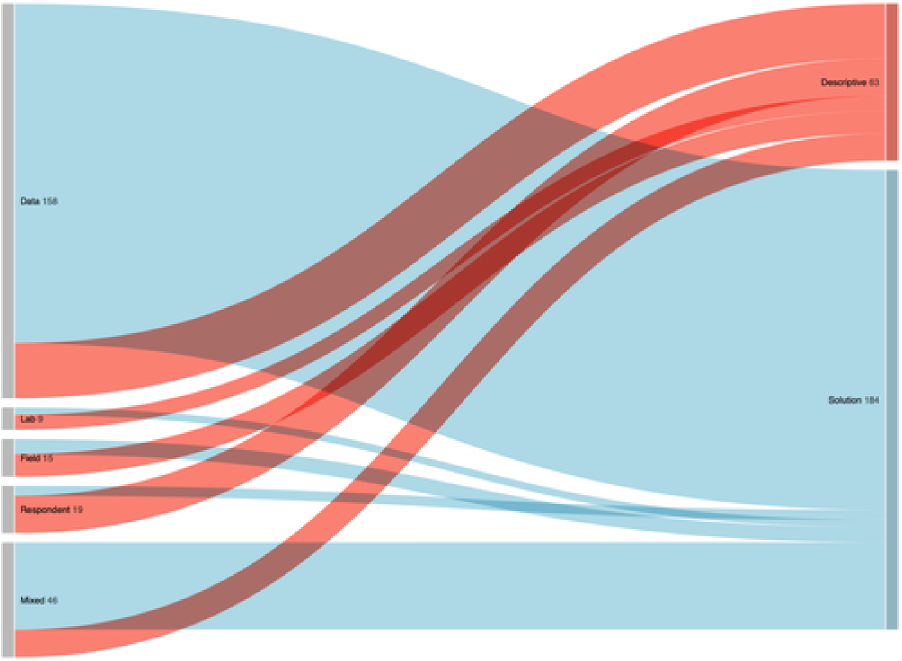

Fig. 5 Mapping research strategy to research contribution. Numbers reflect totals.

5 Implications of Our Findings

We discuss and interpret our findings in this section. Our investigation of ICSE papers and how they address human and social aspects leads to a number of valuable insights, each of which has several implications for the individual researcher as well as for the SE research community.

5.1 Preference for Data Strategies

Why are data strategies commonly used in software engineering research? In recent years, we find easier access to software repositories and diverse data sources concerning software projects. These data sources encompass a rich resource concerning human and technical aspects for conducting empirical software engineering research. We have access to open or proprietary source code when research collaborations are in place; development data such as issues, bugs, test results and commits; crowd sourced community knowledge repositories such as Stack Overflow and Hacker News; and operational and telemetry data from the field such as feature usage, A/B testing results, logging data and user feedback [10]. Analyzing this data is natural for many computer scientists who are trained in the scientific method, as well as in math and statistics, data mining, natural language processing and more recently AI techniques. Indeed, the emergence of new machine learning and AI techniques that scale, may also help explain why data strategies are used so frequently in our field.

Data strategies also show great potential for high precision over the data that is used in these studies (see Fig. 3). The ICSE call for papers highlights the importance of “replication” as an evaluation criterion. Data strategy papers are often accompanied by replication packages that include the software artifact data, algorithms, and tools used in the studies. Other strategies also provide replication packages, but non-data strategies rely more on recruiting human participants to reproduce the data, but at the risk that additional variables may be introduced (e.g., developers may change their opinion or develop new skills over time and the results from such studies may not be easy to replicate). These packages are recognized by the community as suitable and having high potential for replication. Some evidence suggests that replication packages and open science leads to more citations [5], which might also motivate this strategy choice.

Another possible reason for the predominant use of data strategies may be that this type of paper is already common in more recent years of ICSE, and thus new researchers may consider these papers are the expected type of research to submit to this venue. We surveyed the authors of our set of ICSE papers and report the results of this survey in Williams' thesis [3][6]. The focus of this survey was to validate our categorization (which it did), but also to find out why researchers chose certain strategies in their research. As one respondent in the survey reported, “We took the standard approach that would typically be reported in a [topic] conference.” But aiming for generalizability may also be a reason for using data strategies, as many of the author survey participants mentioned. When open source projects are the focus of study, repositories such as GitHub can support research that aims to generalize to a broad set of projects.

However, there are limitations to data strategies as the research strategy framework shows in Fig. 3: data studies do not involve developers directly, and so control over human behaviour is not possible. Indeed, descriptive research aims to capture human and social aspects; it is well known that data alone does not tell the whole story [2, 11] and any conclusions drawn may need to be corroborated with other methods (such as interviews, surveys or observations). In the case of evaluating interventions or technical solutions aimed for developer (or other) stakeholders to use, one can experiment with the data to see if a new technique may offer an improvement. However, such evaluations assume that future developers will use the new intervention in exactly the way the previous intervention was used.

One could argue that a preference for data strategies is to be expected; after all, we should expect to see solutions proposed in a design science study, and data studies are a useful way to validate tool innovations. At the same time, as we argued earlier, software is a socio-technical system and this means

[6] Williams changed her surname following acceptance of her thesis. We do not include the survey as part of this paper, as the paper is already quite long, but we do include the details from the survey as part of our replication package for the interested reader

humans are significantly involved. As we classified papers, this made us wonder how many data papers mention potential stakeholders and implicitly aim at supporting technological rules that will capture how an intervention may lead to a desired effect in a given stakeholder context. That is, even though these papers do not use human-centric research strategies, they nonetheless see the need for their tool or results to be of use or relevant to humans. We consider this next.

5.2 Data Strategies and Human Aspects

How many data strategy papers aim to address social and human aspects? Some data strategy papers may focus on a system aspect or intervention that is independent from human and social concerns. For these papers and this line of research, we would not expect the use of research strategies that directly involve human subjects. For example, a technique that improves the automated build time of source code may not need insights from human subjects to validate the benefits of the technique (i.e., in general, faster = better code compilation). But many of these papers may target interventions or insights that are (or should be) shown to be actionable in a human stakeholder's context, especially interventions that are proposed to assist developer productivity in some way. For example, a paper which proposes an improved defect prediction technique (the solution design) may claim the new technique will improve a developer's ability to reduce bugs. Such a technique may provide suggestions for the developer to consider. Therefore, we would conclude that a paper with such a claim, stated anywhere in the paper, would target the human developer/tester/user as a beneficiary.

To explore this, we coded the data papers for potential *beneficiary* (of the research result, e.g., tool, process). We coded the beneficiary using four categories: 'human', 'system', 'both', and 'researcher'. 'Human' beneficiary implies the paper sees humans as being the ultimate stakeholders, whereas 'system' papers never mention human stakeholders but only improvements in the tool or process. For example, reporting a 15% decrease in runtime, or a 5% increase in F1 measure, would be system benefits. 'Both' refers to a paper where both human and system beneficiaries exist. 'Researcher' is a special label for papers involving meta studies. For example, Siegmund *et al.* [50] is a paper whose primary results are for other researchers.

Table 1 shows the research track papers, using either only data strategy or data strategy plus another strategy, mapped to beneficiary. We deemed something as involving human beneficiaries (as 'human' or 'both') if the paper contained a statement somewhere referring to human stakeholders. To find this mention, we read each paper, augmenting our reading with keyword searches for 'developer', 'human', 'user', 'tester', 'engineer', 'coder', 'programmer'.

Mention of humans in the paper did not mean an in-depth discussion of how a human might benefit from the tool or process. Some papers only tangentially mentioned that humans might benefit from their approach. We also

Strategy	Beneficiary	Count
Data	Researcher	2
Data	System	46
Data	Both	98
Data	Human	11
Mixed	Researcher	0
Mixed	System	3
Mixed	Both	27
Mixed	Human	8

Table 1 Mapping strategy to claimed beneficiary.

were careful to ensure these mentions were not in reference to other related work.

Mention of humans was encouragingly common. For example, Lin *et al.*'s [45] ICSE 2017 data strategy paper, "Feedback-Based Debugging", mentions that their "approach allows developers to provide feedback on execution steps to localize the fault". By comparison, a paper that only referred to a system as the stakeholder noted that "LibD can better handle multi-package third-party libraries in the presence of name-based obfuscation" (Li *et al.* [44]).

We find that even though some may argue that most data strategies were never intended to directly study humans, researchers still find benefiting human stakeholders to be highly relevant. As Table 1 shows, 49 papers exclusively benefit the system (e.g., improve a tool), while 19 exclusively benefit humans. However, 125 papers claim to benefit both system and human. Thus, 144 of 193 papers could be argued as developing an understanding that their tool, process or procedure is ultimately about human activities. Furthermore, 11 papers, despite claiming humans would benefit from their solution, do not use a research strategy that can truly aid in understanding how humans might benefit. Such a claim may be resolved in other studies by that research team.

5.3 Lack of Human-Oriented Research Strategies

Why do we see few respondent, laboratory and field strategies in SE? These research strategy categories all involve responses from or the direct observation of human subjects, but in different settings. Furthermore, we do not know if this may be due to fewer submissions of these types of papers, or if they are less likely to be accepted. We consider both possibilities below.

For respondent strategies, participants typically provide the research data in a setting of their convenience (e.g., through survey or interview responses), whereas in lab and field strategies, the experimenter normally records the data for further analysis in a contrived laboratory setting or real work setting, respectively. Respondent strategies—in particular, surveys or interviews—are used as a secondary method for 12 data strategy papers in our sample set (see Fig. 4), while lab and field strategies that may have more control over human behaviours are used even less often (6 and 7 respectively). Given that these strategies offer other advantages in terms of generalizability, realism and control of variables relevant when human actors are involved, it is surprising they are not used more often, either as a main or secondary method at the community level.

One reason for this low use across our community may be a lack of expertise by software engineering researchers. Many of these methods call for expertise that is not taught in a typical software engineering or computer science educational program (as compared to, say, sociology and other social sciences). Another reason may be that access to developers (as respondents or informants) may be difficult to achieve. Indeed, one of the ICSE authors we surveyed mentioned, "We also would have conducted a field experiment [...], but we didn't have subjects readily available." (see [3]).

Another possible cause for the low use of laboratory and field studies in particular may be because of their potential lack of generalizability. Reviewers can easily attack poor generalizability as a reason to quickly reject a paper. In the survey we conducted with ICSE authors, some were concerned with their ability to publish such studies. One mentioned, "if an approach is not generalizable, it makes no sense to others", and another mentioned, "that's what reviewers easily criticize" (see [3]).

Another issue that may lead to fewer submissions describing field strategies is that these strategies typically rely on the use of qualitative methods and should be evaluated using quite different criteria than those used to evaluate qualitative research (see our limitations section for an example). Reviewers that expect to see threats to validity, such as external, internal and construct, may find qualitative criteria, such as credibility and transferability, as unacceptable, or at least unfamiliar, due to a different epistemological stance.

Finally, running studies with human subjects requires the additional step (in most academic institutions) of acquiring approval from an ethics review board, and the use of human subjects inevitably introduces other complications (sometimes people do not show up or have unique abilities that impact the study). This step is generally not needed for data strategies, although ethical concerns about the use of some data resources has been raised in the community.

To understand if the above reasons explain why these methods are not used—or at least are not published as often as data strategies in ICSE—requires future work.

5.4 Research Strategy Triangulation and Diversification

In terms of triangulation, Runkel and McGrath [21] recommend the use of various research strategies and data sources to address imbalance in prioritizing generalizability, realism, and control. We also introduced the criterion of precision in our framework to capture the notion that there is the potential for precise measurements of data in the data strategies. Some lab and field experiments and respondent strategies may also have potential for precise measurements over data that is gathered in those studies, but generally data strategies have the highest potential for precision as many rely on retrospectively gathered or simulated data that is not likely to change (there are no humans involved in those studies that may "mess" things up). As we noted previously, data strategies show the lowest potential for control over variables that may come into play when human participants are involved in the study. Hence, we agree with Runkel and McGrath that triangulation is desirable to help achieve a better balance across control and precision, as well as generalizability and realism. While we could not determine whether authors triangulated their work outside of their papers, our findings show that the vast majority of the papers in our sample used a single research strategy (82%).

That said, the responsibility for triangulation does not need to be on the level of individual papers. While it is valuable to triangulate findings with multiple research strategies within a single paper, it may be impractical and many studies have valuable insights that warrant an entire paper. Instead, we suggest that the community as a whole should be responsible for triangulation; it is unreasonable to expect each researcher to have the knowledge and the means to conduct research using all or even several of the strategies available. Thus, we call on the community to reflect on the value of studies that triangulate work conducted by other researchers. Harnessing our strengths and skills with particular methods as individual researchers to triangulate the work of others will help us develop more impactful findings for the community as a whole [16], and will support us to better understand the social as well as technical aspects of software development.

In terms of diversifying strategies, there are other benefits and drawbacks to be considered (in addition to realism, control, generalizability and precision). Field strategies often lead to actionable insights and understanding of why things work the way they do (they contribute descriptive knowledge). They can also lead to new ideas that may not be easy to discern from lab, respondent or data strategies. For example, the understanding of a problem may not be apparent until one goes into the field and studies a specific problem instance, while ideas for new interventions may be derived from watching how developers deal with or work around a problem in the real world. These observations often lead to new insights and innovations. Field strategies are also useful for studying the non-adoption of interventions which cannot be easily understood from other strategies. On the other hand, data strategies may be easier to replicate, while both respondent and data strategies may more easily scale to larger and broader populations.

In sum, multiple research strategies should be used, not just to triangulate specific findings, but to further add to insights concerning the problem instance, the solution design, and the solution evaluation (see Fig. 2). Multiple strategies will help researchers, either individually or at the community level, develop and refine technological rules that capture how tool innovations will lead to desirable effects in particular stakeholder contexts.

5.5 Design science and software engineering

In our previous work, we used the design science lens to categorize research contributions from the ACM SIGSOFT Distinguished Paper Award papers from 2015-2018. We found that 27 of these papers involve the design and/or evaluation of a technical solution, while 8 papers mostly contribute descriptive knowledge. We see a similar balance of descriptive knowledge (63 papers) to solution design papers (184) in the broader cohort of 253 papers we analyzed.

As we found and discuss earlier, many of the solution papers rely on data strategies as their primary research strategy and many do not evaluate the proposed solutions with stakeholders directly and instead rely on retrospectively gathered or simulated data for their evaluation. Design science is a paradigm that promotes the involvement of stakeholders as part of the research methodology and thus using design science in software engineering may lead to more attention placed on understanding real world problem instances software engineering stakeholders face, and designing and evaluating solutions for and with those stakeholders. Doing so, may lead to research that is more directly applicable to industry contexts and result in technological rules that other researchers can refine and extend in their own research.

6 Limitations

In this section, we identify the limitations associated with this research and the measures we took to mitigate these issues through the research design. We use the Roller and Lavrakas [19] "Total Quality Framework" for qualitative research, consisting of the subdomains "Credibility" (of the data collection), "Analyzability" (of data analysis), "Transparency" (of the reporting), and "Usefulness" (of the results). See Fig. 6.

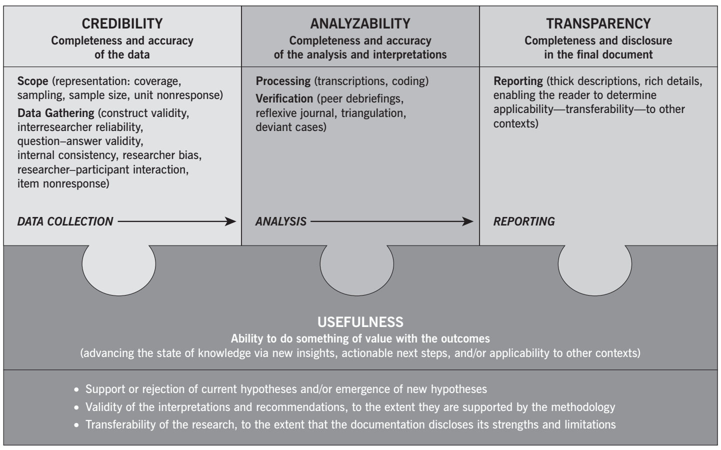

Fig. 6 The *Total Quality Framework* of [19].

6.1 Credibility

Credibility is an assessment of the completeness and accuracy of the data. The **scope** of our study was limited to ICSE papers from 2015-2017. Different years, different venues, and different tracks may have produced a different distribution of research strategy and data source use, and may have attracted researchers with different experiences and opinions to participate in our survey. As mentioned earlier, there are other venues that clearly focus on human aspects, including the CHASE workshop, co-located with ICSE since 2011, as well as other venues such as VL/HCC[7] and CSCW[8]. Thus we recognize our findings are particular to ICSE, but we feel important to share as the ICSE publishing venue is recognized as being inclusive in terms of topics and methods, but it is also seen by many as the premier publishing venue in software engineering.

Another aspect of credibility is the **data gathering** process. Because many of the analysis tasks rely on human judgment, such as open coding and classification of research papers, heavily relying on a single researcher introduces the potential for researcher bias. To mitigate this issue, we validated the classifications from the categorization study by having the authors of papers classify their own work as part of a survey which is reported in [3]. We also validated our thematic coding of paper contributions and beneficiaries with a second independent coder, and provided a number of analysis documents on our website to make the analysis process as transparent and traceable as possible. We found very low disagreement between the coders once we had conducted a debriefing session.

The authors derive from a predominantly human-centered software research background, and so this may have influenced our interpretation of papers outside our area of expertise (e.g., for areas such as automated testing). To mitigate this we relied on the authors themselves to inform us of the intended purpose of the study, and attempted to be as expansive as possible when interpreting the data.

We assigned each paper a single contribution type for the research study (i.e., either descriptive or solution), but this is a coarse description of a single paper's epistemological goals. For example, we coded the main contribution type, but some studies might do exploration, then design a tool. A richer description would leverage design science terminology more fully, such as in Engström *et al.* [9].

6.2 Analyzability

Analyzability is an assessment of the accuracy of the paper's analysis and interpretations. To mitigate issues that may arise based on our quadrant model, we triangulated the findings from our categorization study by member checking

[7] `http://conferences.computer.org/VLHCC/`
[8] ACM Conference on Computer Supported Cooperative Work, `https://cscw.acm.org`

with authors without relying on the use of the terminology from the research lens to determine a common ground between the authors and the researchers. We met regularly to debrief one another as to our coding approach, and flagged contentious papers (i.e., where coders disagreed) in order to resolve disagreements. We relied on an online spreadsheet to host our coding data (available as a replication package), and created an R script to process and calculate our descriptive results.

6.3 Transparency

Transparency refers to the way in which we have presented our results. We rely mainly on the replication package to ensure transparency. We can also rely, to some extent, on the familiarity readers likely have with the domain. Since we write about software research, we did not see a need to provide detailed descriptions of the domain.

6.4 Usefulness

We have presented two research mechanisms for evaluating socio-technical contributions in software research—a research strategy framework and a simplified version of an existing research contribution lens (based on design science). We believe these are ready to apply to other venues (e.g., other software engineering journals or conferences). To facilitate this further application, we provide a number of documents on our supplementary website designed to help other researchers follow our methodology. We welcome replication studies—or indeed, triangulation studies with new research strategies—on additional years of ICSE and other venues to explore the differences that may exist between venues and time periods in software engineering research.

7 Conclusions

Understanding the complexities of human behavior requires the use of diverse research strategies and both active and inactive forms of human participation. However, through our categorization study of ICSE technical track papers from 2015 to 2017, we found a skew towards data strategies across our community. We do not suggest that any one data strategy paper should use different or triangulate using additional methods as data strategies may make sense, particularly in the early stages of a research program. However, at a community level, we may wish to consider and reflect on this imbalance, and if such an imbalance should persist over time for specific topics.

Since much of our research is aimed at human and social aspects of software development, it may make sense to reflect on how we formulate our call-for-papers and on how we evaluate our ICSE papers. This may help us to understand if either of these aspects may contribute to finding fewer papers that directly study human aspects appearing in the ICSE main track. For example, realism may help improve the impact of our research on industry, but at the expense of generalizability and precision over data measurements. Reflecting on these tradeoffs is something authors and reviewers may need to do.

The framework we describe in Section 3.1 may be used by other researchers to reflect on the implications of their personal choice of methods, in terms of generalizability, realism and control of human aspects, as well as by other researchers that may wish to reflect on a body of work in other venues, and possibly compare with the ICSE technical track. Our hope is that by reflecting on the type of research we do and the research strategies we use in our community, our research will have more impact both on researchers and on industry.

References

1. van Aken, J.E.: Management research based on the paradigm of the design sciences: The quest for field-tested and grounded technological rules. Journal of Management Studies **41**(2), 219–246 (2004). DOI 10.1111/j.1467-6486.2004.00430.x
2. Aranda, J., Venolia, G.: The secret life of bugs: Going past the errors and omissions in software repositories. In: icse, pp. 298–308 (2009). DOI 10.1109/ICSE.2009.5070530
3. Bornholdt, C.: Methodology matters: mapping software engineering research through a sociotechnical lens. Master's thesis, University of Victoria (2019). URL `https://dspace.library.uvic.ca//handle/1828/9997`
4. Brooks Jr., F.P.: The Mythical Man-month (Anniversary Ed.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1995)
5. Colavizza, G., Hrynaszkiewicz, I., Staden, I., Whitaker, K., McGillivray, B.: The citation advantage of linking publications to research data. Tech. Rep. arXiv:1907.02565, arXiv (2019). URL `https://arxiv.org/abs/1907.02565`
6. Conradi, R., Wang, A.I.: Empirical Methods and Studies in Software Engineering: Experiences from Esernet. Springer-Verlag, Berlin, Heidelberg (2003)
7. DeMarco, T., Lister, T.: Peopleware: Productive Projects and Teams. Dorset House Publishing Co., Inc., New York, NY, USA (1987)
8. Easterbrook, S., Singer, J., Storey, M.A., Damian, D.: Selecting Empirical Methods for Software Engineering Research, pp. 285–311. Springer London, London (2008). DOI 10.1007/978-1-84800-044-5_11
9. Engström, E., Storey, M.D., Runeson, P., Höst, M., Baldassarre, M.T.: A review of software engineering research from a design science perspective. CoRR **abs/1904.12742** (2019). URL `http://arxiv.org/abs/1904.12742`
10. Hassan, A.E.: The road ahead for mining software repositories. In: 2008 Frontiers of Software Maintenance, pp. 48–57 (2008). DOI 10.1109/FOSM.2008.4659248
11. Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D.M., Damian, D.: The promises and perils of mining GitHub. In: International Working Conference on Mining Software Repositories (2014). DOI 10.1145/2597073.2597074
12. Kitchenham, B.: Empirical Paradigm – The Role of Experiments, pp. 25–32. Springer Berlin Heidelberg, Berlin, Heidelberg (2007). DOI 10.1007/978-3-540-71301-2_9
13. Kitchenham, B.A., Pfleeger, S.L.: Personal Opinion Surveys, pp. 63–92. Springer London, London (2008). DOI 10.1007/978-1-84800-044-5_3
14. Ko, A.J., LaToza, T.D., Burnett, M.M.: A practical guide to controlled experiments of software engineering tools with human participants. Empirical Software Engineering **20**(1), 110–141 (2015). DOI 10.1007/s10664-013-9279-3
15. Kontio, J., Bragge, J., Lehtola, L.: The Focus Group Method as an Empirical Tool in Software Engineering, pp. 93–116. Springer London, London (2008). DOI 10.1007/978-1-84800-044-5_4

16. Mackay, W.E., Fayard, A.L.: HCI, natural science and design: A framework for triangulation across disciplines. In: International Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques, pp. 223–234 (1997). DOI 10.1145/263552.263612
17. McDermid, J.A., Bennett, K.H.: Software engineering research: a critical appraisal. IEE Proceedings-Software **146**(4), 179–186 (1999)
18. McGrath, J.E.: Methodology matters: Doing research in the behavioral and social sciences. In: R.M. Baecker, J. Grudin, W. Buxton, S. Greenberg (eds.) Readings in Human-Computer Interaction: Toward the Year 2000, pp. 152–169. Morgan Kaufmann Publishers Inc. (1995)
19. Roller, M.R., Lavrakas, P.J.: Applied Qualitative Research Design: A Total Quality Framework Approach. Guilford Press (2015). URL `https://www.amazon.com/Applied-Qualitative-Research-Design-Framework/dp/1462515754`
20. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. Empirical Software Engineering **14**(2), 131 (2008). DOI 10.1007/s10664-008-9102-8
21. Runkel, P.J., McGrath, J.E.: Research on Human Behavior. Holt, Rinehart, and Winston, Inc. (1972)
22. Seaman, C.B.: Qualitative Methods, pp. 35–62. Springer London, London (2008). DOI 10.1007/978-1-84800-044-5_2
23. Sharp, H., Dittrich, Y., de Souza, C.R.B.: The role of ethnographic studies in empirical software engineering. IEEE Transactions on Software Engineering **42**(8), 786–804 (2016). DOI 10.1109/TSE.2016.2519887
24. Sharp, H., Robinson, H.: Some social factors of software engineering: the maverick, community and technical practices. In: ACM SIGSOFT Software Engineering Notes, vol. 30, pp. 1–6. ACM (2005)
25. Shaw, M.: Writing good software engineering research papers: Minitutorial. In: Proceedings of the 25th International Conference on Software Engineering, ICSE '03, pp. 726–736. IEEE Computer Society, Washington, DC, USA (2003)
26. Shneiderman, B.: Software Psychology: Human Factors in Computer and Information Systems (Winthrop Computer Systems Series). Winthrop Publishers (1980)
27. Shull, F., Singer, J., Sjøberg, D.I.: Guide to Advanced Empirical Software Engineering. Springer-Verlag, Berlin, Heidelberg (2008)
28. Singer, J., Sim, S.E., Lethbridge, T.C.: Software Engineering Data Collection for Field Studies, pp. 9–34. Springer London, London (2008). DOI 10.1007/978-1-84800-044-5_1
29. Sjøberg, D.I.K., Dybå, T., Anda, B.C.D., Hannay, J.E.: Building Theories in Software Engineering, pp. 312–336. Springer London, London (2008). DOI 10.1007/978-1-84800-044-5_12
30. Sjøberg, D.I.K., Dyba, T., Jorgensen, M.: The future of empirical methods in software engineering research. In: Future of Software Engineering, pp. 358–378 (2007). DOI 10.1109/FOSE.2007.30
31. Stol, K.J., Fitzgerald, B.: The abc of software engineering research. ACM Trans. Softw. Eng. Methodol. **27**(3), 11:1–11:51 (2018). DOI 10.1145/3241743. URL `http://doi.acm.org/10.1145/3241743`
32. Theisen, C., Dunaiski, M., Williams, L., Visser, W.: Writing good software engineering research papers: Revisited. In: 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C), pp. 402–402 (2017). DOI 10.1109/ICSE-C.2017.51
33. Vinson, N.G., Singer, J.: A Practical Guide to Ethical Research Involving Humans, pp. 229–256. Springer London, London (2008). DOI 10.1007/978-1-84800-044-5_9
34. Weinberg, G.M.: The Psychology of Computer Programming. John Wiley & Sons, Inc., New York, NY, USA (1985)
35. Whitworth, B.: The social requirements of technical systems. In: B. Whitworth, A. de Moor (eds.) Handbook of Research on Socio-Technical Design and Social Networking Systems, pp. 2–22. IGI Global (2009). DOI 10.4018/978-1-60566-264-0
36. Wohlin, C., Höst, M., Henningsson, K.: Empirical Research Methods in Software Engineering, pp. 7–23. Springer Berlin Heidelberg, Berlin, Heidelberg (2003). DOI 10.1007/978-3-540-45143-3_2

37. Wohlin, C., Runeson, P., Host, M., Ohlsson, M.C., Regnell, B., Wessln, A.: Experimentation in Software Engineering. Springer Publishing Company, Incorporated (2012)
38. Zelkowitz, M.V.: Techniques for Empirical Validation, pp. 4–9. Springer Berlin Heidelberg, Berlin, Heidelberg (2007). DOI 10.1007/978-3-540-71301-2_2

References to papers within the study sample

39. Christakis, M., Emmisberger, P., Godefroid, P., Müller, P.: A general framework for dynamic stub injection. In: Proceedings of the ACM/IEEE International Conference on Software Engineering, pp. 586–596 (2017). DOI 10.1109/ICSE.2017.60. URL https://doi.org/10.1109/ICSE.2017.60
40. Faitelson, D., Tyszberowicz, S.: Uml diagram refinement (focusing on class- and use case diagrams). In: Proceedings of the ACM/IEEE International Conference on Software Engineering, pp. 735–745 (2017). DOI 10.1109/ICSE.2017.73. URL https://doi.org/10.1109/ICSE.2017.73
41. Hoda, R., Noble, J.: Becoming agile: A grounded theory of agile transitions in practice. In: Proceedings of the ACM/IEEE International Conference on Software Engineering, pp. 141–151 (2017). DOI 10.1109/ICSE.2017.21. URL https://doi.org/10.1109/ICSE.2017.21
42. Joblin, M., Apel, S., Hunsen, C., Mauerer, W.: Classifying developers into core and peripheral: An empirical study on count and network metrics. In: Proceedings of the ACM/IEEE International Conference on Software Engineering, pp. 164–174 (2017). DOI 10.1109/ICSE.2017.23. URL https://doi.org/10.1109/ICSE.2017.23
43. Kononenko, O., Baysal, O., Godfrey, M.W.: Code review quality: How developers see it. In: Proceedings of the ACM/IEEE International Conference on Software Engineering, pp. 1028–1038 (2016). DOI 10.1145/2884781.2884840. URL http://doi.acm.org/10.1145/2884781.2884840
44. Li, M., Wang, W., Wang, P., Wang, S., Wu, D., Liu, J., Xue, R., Huo, W.: LibD: Scalable and precise third-party library detection in android markets. In: Proceedings of the ACM/IEEE International Conference on Software Engineering (2017). DOI 10.1109/icse.2017.38
45. Lin, Y., Sun, J., Xue, Y., Liu, Y., Dong, J.: Feedback-based debugging. In: Proceedings of the ACM/IEEE International Conference on Software Engineering (2017). DOI 10.1109/icse.2017.43. URL https://doi.org/10.1109/icse.2017.43
46. Nistor, A., Chang, P.C., Radoi, C., Lu, S.: CARAMEL: Detecting and fixing performance problems that have non-intrusive fixes. In: Proceedings of the ACM/IEEE International Conference on Software Engineering (2015). DOI 10.1109/icse.2015.100. URL https://doi.org/10.1109/icse.2015.100
47. Prechelt, L., Schmeisky, H., Zieris, F.: Quality experience: A grounded theory of successful agile projects without dedicated testers. In: Proceedings of the ACM/IEEE International Conference on Software Engineering, pp. 1017–1027 (2016). DOI 10.1145/2884781.2884789. URL http://doi.acm.org/10.1145/2884781.2884789
48. Sadowski, C., van Gogh, J., Jaspan, C., Söderberg, E., Winter, C.: Tricorder: Building a program analysis ecosystem. In: Proceedings of the ACM/IEEE International Conference on Software Engineering, pp. 598–608 (2015). URL http://dl.acm.org/citation.cfm?id=2818754.2818828
49. Sedano, T., Ralph, P., Peraire, C.: Software development waste. In: Proceedings of the ACM/IEEE International Conference on Software Engineering (2017). DOI 10.1109/icse.2017.20. URL http://dx.doi.org/10.1109/ICSE.2017.20
50. Siegmund, J., Siegmund, N., Apel, S.: Views on internal and external validity in empirical software engineering. In: Proceedings of the ACM/IEEE International Conference on Software Engineering, vol. 1, pp. 9–19. IEEE (2015)
51. Stol, K.J., Ralph, P., Fitzgerald, B.: Grounded theory in software engineering research: A critical review and guidelines. In: Proceedings of the ACM/IEEE International Conference on Software Engineering, pp. 120–131 (2016). DOI 10.1145/2884781.2884833

52. Treude, C., Robillard, M.P.: Augmenting api documentation with insights from stack overflow. In: Proceedings of the ACM/IEEE International Conference on Software Engineering, pp. 392–403 (2016). DOI 10.1145/2884781.2884800. URL `http://doi.acm.org/10.1145/2884781.2884800`
53. Vakilian, M., Phaosawasdi, A., Ernst, M.D., Johnson, R.E.: Cascade: A universal programmer-assisted type qualifier inference tool. In: Proceedings of the ACM/IEEE International Conference on Software Engineering, pp. 234–245 (2015). URL `http://dl.acm.org/citation.cfm?id=2818754.2818785`