Empirical Software Engineering (2020) 25:2630–2660
https://doi.org/10.1007/s10664-020-09818-7

How software engineering research aligns with design science: a review

Emelie Engström[1] · **Margaret-Anne Storey**[2] · **Per Runeson**[1] · **Martin Höst**[1] · **Maria Teresa Baldassarre**[3]

Published online: 18 April 2020

Abstract

Background Assessing and communicating software engineering research can be challenging. Design science is recognized as an appropriate research paradigm for applied research, but is rarely explicitly used as a way to present planned or achieved research contributions in software engineering. Applying the design science lens to software engineering research may improve the assessment and communication of research contributions.

Aim The aim of this study is 1) to understand whether the design science lens helps summarize and assess software engineering research contributions, and 2) to characterize different types of design science contributions in the software engineering literature.

Method In previous research, we developed a visual abstract template, summarizing the core constructs of the design science paradigm. In this study, we use this template in a review of a set of 38 award winning software engineering publications to extract, analyze and characterize their design science contributions.

Results We identified five clusters of papers, classifying them according to their different types of design science contributions.

Conclusions The design science lens helps emphasize the theoretical contribution of research output—in terms of technological rules—and reflect on the practical relevance, novelty and rigor of the rules proposed by the research.

Keywords Design science · Research review · Empirical software engineering

1 Introduction

Design science is a paradigm for conducting and communicating applied research such as software engineering. The goal of design science research is to produce prescriptive

Communicated by: Sven Apel

✉ Emelie Engström
emelie.engstrom@cs.lth.se

Extended author information available on the last page of the article.

knowledge for professionals in a discipline and to share empirical insights gained from investigations of the prescriptions applied in context (van Aken 2004). Such knowledge is referred to as "design knowledge" as it helps practitioners design solutions to their problems. Similar to other design sciences, much software engineering research aims to design solutions to practical problems in a real-world context.

Design science is an established research paradigm in the fields of information systems (Hevner et al. 2004) and other engineering disciplines, such as mechanical, civil, architectural, and manufacturing engineering.[1] It is also increasingly used in computer science; for example, it is now accepted as the *de facto* paradigm for presenting design contributions from information visualization research (Sedlmair et al. 2012). Although Wierenga has promoted design science for capturing design knowledge in software engineering (Wieringa 2014), we seldom see it being referred to in our field (although there are some exceptions (Wohlin and Aurum 2015)). We are puzzled by its low adoption as the use of this lens could increase the clarity of research contributions for both practitioners and researchers, as it has been shown to do in other fields (Shneiderman 2016).

The goal of our research is to investigate if and how a design science paradigm may be a viable way to assess and communicate research contributions in existing software engineering literature. To this end, we consider a set of software engineering research papers and view these contributions through a design science lens by using and improving a visual abstract template we previously developed to showcase design knowledge (Storey et al. 2017).

We inspected 38 ACM distinguished papers published at the International Conference on Software Engineering (ICSE) over a five-year period—publications considered by many in the community as well-known exemplars of fine software engineering research, and papers that are expected to broadly represent the diverse topics addressed by our research community. Although these papers set a high bar for framing their research contributions, we found that the design science lens improved our understanding of their contributions. Also, most of the papers described research contributions that are congruent with a design science paradigm, even though none of them explicitly used the term. Applying this lens helped us elucidate certain aspects of the contributions (such as relevance, novelty and rigor), which in some cases were obscured by the original framing of the paper. However, not all the papers we considered produced design knowledge, thus some research publications do not benefit from using this lens.

Our analysis from this exercise led to five clusters of papers based on the type of design knowledge reported. We compare the papers within each cluster and reflect on how the design knowledge is typically achieved and reported in these clusters of papers.

In the remainder of this paper, we first present background on design science and our conceptualization of it by means of a visual abstract template (Section 2). We then describe our methodology for generating visual abstracts for the cohort of ACM distinguished papers we studied (Section 3), and use the information highlighted by the abstracts to extract the design knowledge in each paper. Finally, we cluster the papers by the type of design knowledge produced (Section 4). We interpret and discuss the implications of our findings (Sections 5 and 6), outline the limitations of our study (Section 7), and discuss related work (Section 8) before concluding the paper (Section 9).

[1] springer.com/journal/163 Research in Engineering Design

2 Background

Our conceptualization of design science in software engineering, which our analysis is based on, was formed from a thorough review of the literature and a series of internal group workshops on the topic. This work helped us develop a visual abstract template to use as a lens for communicating and assessing research contributions (Storey et al. 2017). In this section, we give a brief introduction to design science and the visual abstract template. We use the term *design knowledge* to refer to the knowledge produced in design science research.

2.1 Design Science

Design science is a common research paradigm used in many fields of information systems and other engineering disciplines. By research paradigm, we refer to van Aken's definition: "the combination of research questions asked, the research methodologies allowed to answer them and the nature of the pursued research products" (van Aken 2005). The mission of design science is to solve real-world problems. Hence, design science researchers aim to develop general design knowledge in a specific field to help practitioners create solutions to their problems. In Fig. 1, we illustrate the relationship between the *problem domain* and *solution domain*, as well as between *theory* and *practice*. The arrows in the figure represent different types of contributions of design science research, i.e., problem conceptualization, solution design, instantiation, abstraction, and validation.

Design knowledge is holistic and heuristic by its nature, and must be justified by in-context validations (Wieringa 2014; van Aken 2004). The term holistic is used by van Aken (2004) and refers to the "magic" aspect of design knowledge, implying that we never fully understand why a certain solution works in a specific context. There will always be hidden context factors that affect a problem-solution pair (Dybå et al. 2012). As a consequence, we can never prove the effect of a solution conclusively, and must rely on heuristic prescriptions. By evaluating multiple problem-solution pairs matching a given prescription, our understanding about that prescription increases. Design knowledge can be expressed in terms of *technological rules* (van Aken 2004), which are rules that capture general knowledge about the *mappings* between *problems* and proposed *solutions*.

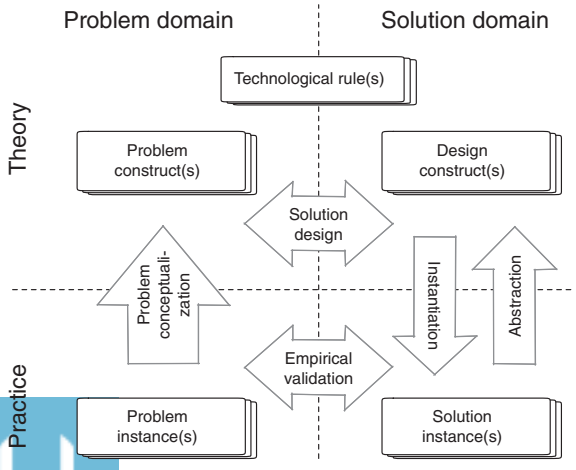

Fig. 1 An illustration of the interplay between problem and solution as well as between theory and practice in design science research. The arrows illustrate the knowledge-creating activities, and the boxes represent the levels and types of knowledge that is created

Van Aken describes the typical design science research strategy to be the multiple case study (van Aken 2004), which can be compared with alpha and beta testing in clinical research, i.e., first case and succeeding cases. Rather than proving theory, design science research strives to refine theory, i.e., finding answers to questions about why, when, and where a solution may or may not work. Each new case adds insights that can refine the technological rule until saturation is achieved (van Aken 2004). Gregor and Hevner present a similar view of knowledge growth through multiple design cycles (Gregor and Hevner 2013). Wieringa and Moralı (2012) and Johannesson and Perjons (2014) discuss action research as one of several empirical methodologies that can be used to produce design knowledge. Sein et al. (2011) propose how design science can be adapted by action research to emphasise the construction of artifacts in design science. However, action research does not explicitly aim to develop knowledge that can be transferred to other contexts, but rather it tries to make a change in one specific local context.

2.2 A Design Science Visual Abstract Template

The visual abstract template we designed, shown in Fig. 2, captures three main aspects of design science contributions: 1) the theory proposed or refined in terms of a technological rule; 2) the empirical contribution of the study in terms of one or more instances of a problem-solution pair and the corresponding design and validation cycles; and 3) support for the assessment of the value of the produced knowledge in terms of relevance, rigor, and novelty. While adhering to the design science paradigm puts the focus on how to *produce*

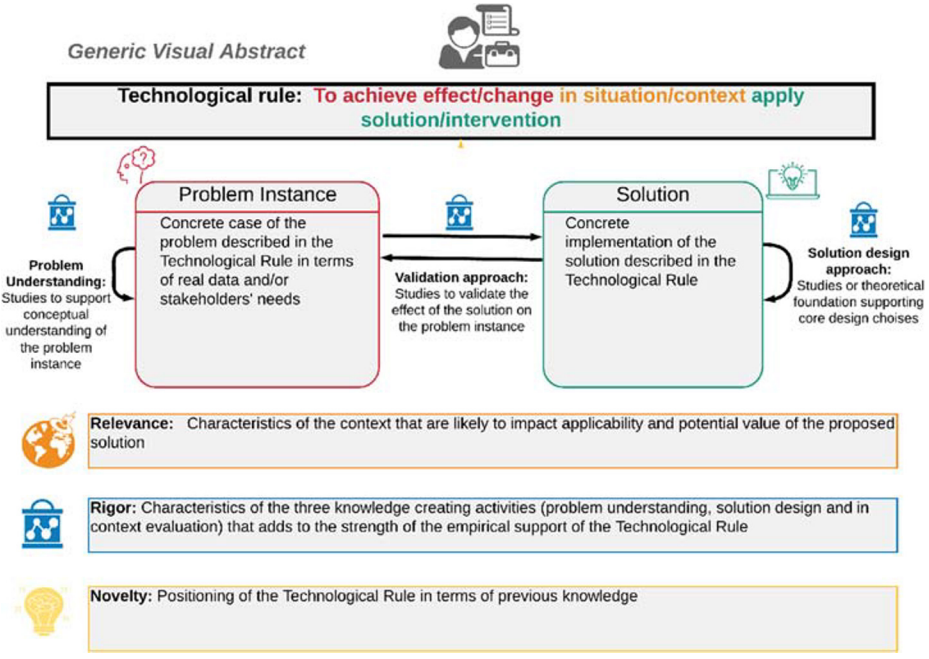

Fig. 2 The visual abstract template (Storey et al. 2017) capturing 1) the theory proposed or refined in terms of a technological rule; 2) the empirical contribution of the study in terms of a problem-solution instance and the corresponding design and validation cycles; and 3) support for the assessment of the value of the produced knowledge in terms of relevance, rigor, and novelty

and *assess* design knowledge (i.e., the technological rules), our visual abstract template is designed to help researchers effectively *communicate* as well as justify design knowledge. It also helps highlight which instantiations of the rule have been studied and how they were validated, how problem understanding was achieved, and what foundations for the proposed solution were considered. In the visual abstract template, the researcher is encouraged to reflect on how a study adds new knowledge to the general theory (i.e., the constructs of the technological rule) and to be aware of the relationship between the general rule and its instantiation (the studied problem-solution pair).

2.2.1 The Technological Rule

In line with van Aken (2004), our visual abstract template emphasizes technological rules (the top box in Fig. 2) as the main takeaway of design science within software engineering research. A technological rule can be expressed in the form: *To achieve <Effect > in <Situation > apply <Intervention>*. Here, a class of software engineering problems is generalized to a stakeholder's desired effect of applying a potential intervention in a specified situation. Making this problem generalization explicit helps the researcher identify and communicate the different value-creating aspects of a research study or program. Refinements or evaluation of the technological rule may be derived from any one of the three processes of *problem conceptualization*, *solution design*, or *empirical validation*, applied in each instantiation.

Technological rules can be expressed at any convenient abstraction level and be hierarchically related to each other. However, technological rules expressed at a high abstraction level (e.g., "to produce software of high quality, apply good software engineering practices") tend to be either too high-level or too bold (easy to debunk). On the other hand, an abstraction level that is too low level may lead to narrowly scoped and detailed rules that may lack relevance for most software engineers. It is important to explicitly formulate the technological rule when presenting design science research and to be consistent with it both when arguing for its relevance and novelty, as well as when presenting the empirical (or analytical) support for the claims.

2.2.2 The Problem-Solution Pair

The main body of the visual abstract template (the middle section in Fig. 2) focuses on the empirical contribution of one or more studies, and is composed of two boxes for the problem-solution instantiation of the technological rule and three corresponding descriptions of the knowledge-creating activities, problem conceptualization, solution design, and validation. This part of the VA helps to distinguish which empirical studies are done and how they contribute to insights about the problem, the solution, or solution evaluation.

2.2.3 The Assessment Criteria

The ultimate goal of design science research is to produce general design knowledge rather than to solve the problems of unique instances. Thus, the value of the research should be assessed with respect to the technological rule (i.e., the design knowledge) produced. The information in the three assessment boxes (the bottom of Fig. 2) aims to help the reader make an assessment that is relevant for their context. Hevner presents three research cycles in the conceptual model of design science, namely the *relevance*, *rigor*, and *design*

cycles (Hevner et al. 2004). We propose that the contributions of design science research be assessed accordingly with respect to *relevance, rigor*, and *novelty*.

The *relevance* box aims to support answering the question *To whom is this technological rule relevant?* Relevance is a subjective concept and we are not striving to find a general definition. Instead we suggest that the basic information needed to assess relevance of a research contribution is the potential effect of the proposed intervention combined with the addressed context factors. The relevance of a research contribution could be viewed from two perspectives: the targeted practitioner's perspective, and the research community's perspective. From the individual practitioner's point of view, the relevance of a research contribution relates to the prevalence and severity of the addressed problem and the applicability of the proposed intervention. This can be assessed by comparing their specific context with the one described in the research report. For the research community, a measure of relevance often relates to how common or severe the studied problem is. To enable both types of assessments, relevant context factors need to be reported.

The *rigor* box aims to support answering the question *How mature is the technological rule?* Rigor of a design science study refers to the strength of the added support for the technological rule and may be assessed with respect to all of the three knowledge-creating activities: problem conceptualization, solution design, and empirical validation. However, solution design is a creative process that does not necessarily add to the rigor of a study. One aspect of rigor in the design activity could be the extent to which the design is built on prior design knowledge. Also, the consideration of alternative solutions could be taken into account. The other two activities—problem conceptualization and empirical validation—are based on common empirical methods on which relevant validity criteria (e.g., construct validity) can be applied. Note that the template only captures the claims made in the paper, and the validity of the claims are assumed to be assessed in the peer review process.

The *novelty* box aims to capture the positioning of the technological rule in terms of previous knowledge, and it supports answering the question *Are there other comparable rules (similar, more precise, or more general rules) that should also be considered when designing a similar solution in another context?* Technological rules may be expressed at several abstraction levels; thus, it is always possible to identify a lower abstraction level where a research contribution may be novel, but doing so may be at the cost of more general relevance. For example, a technological rule that expresses the efficiency of a technique in general may be made more specialized if it instead expresses the efficiency in one specific type of project that has been studied. Then the relevance is less general, and the novelty may be increased since it is the first investigation at that level of detail. Similarly, rigor is increased since the claims are less bold.

To optimize rigor, novelty, and relevance of reported research, the researcher should strive to express the technological rule at the highest useful abstraction level, i.e., a level at which it is novel, the provided evidence gives strong support and it is not debunked by previous studies (or common sense). However, adding empirical support for existing but under-evaluated technological rules has value, making novelty less important than the rigor and relevance criteria. To this extent, replication of experiments has been discussed (Carver et al. 2014; Juristo and Gómez 2010; Shull et al. 2008) and is encouraged by the software engineering community.[2] The incremental adding of empirical support for a technological

[2]See e.g. https://2018.fseconference.org/track/rosefest-2018

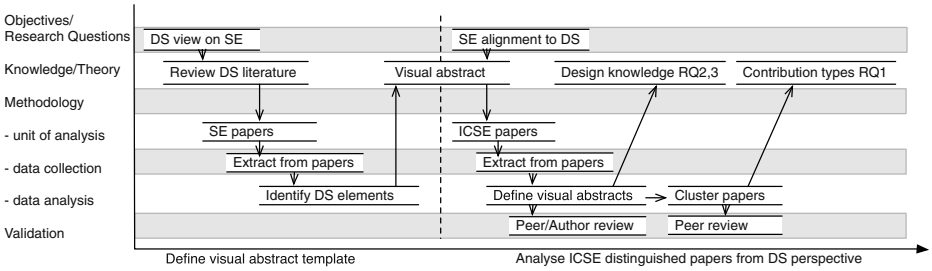

Fig. 3 The approach followed to develop the initial version of the visual abstract (the left side) and the main steps of the research presented in this paper (the right side)

rule could be referred to as conceptual replication in which the same research question is evaluated by using a different study design, as discussed by Shull et al. (2008).

3 Methodology

The main goal of this work was to investigate how well software engineering (SE) research contributions are aligned with a design science paradigm. As part of this work, we aimed to answer the following *research questions*:

RQ1 From a design science perspective, what *types of contributions* do we find in the SE community?

RQ2 In papers that present design knowledge, how clearly are the theoretical contributions (i.e., the *technological rules*) defined in these papers?

RQ3 How are *novelty*, *relevance* and *rigor* discussed in papers with design knowledge contributions?

As mentioned above, our earlier research produced a visual abstract template for communicating design science research in SE (Storey et al. 2017). The principal steps of that study are presented in the left part of Fig. 3. We started by *reviewing the design science literature* and extracting elements of the paradigm from different authors (van Aken 2004; Hevner et al. 2004; Wieringa 2014), from which we created our initial conceptualization of design science as a paradigm for SE. We then studied *SE papers* with design science contributions, and *extracted* information related to such contributions from the papers iteratively to *identify elements of design science*. This work resulted in the visual abstract template, with examples of SE research (Storey et al. 2017).

In this paper we describe how we use the visual abstract template to describe the research contributions in a particular set of papers from the main technical track at the ICSE conference: those that were selected as the top 10% of papers (i.e., papers designated the ACM SIGSOFT Distinguished Paper Award[3]) across five years of the conference (2014–2018 inclusive) We chose ICSE because it is considered to be one of the top publishing venues in software engineering that covers a broad set of diverse topics, and we chose the "best" of those papers because we expected this cohort would represent exemplars of fine research. In total, we considered and applied the visual abstract template to describe the research

[3] https://www.sigsoft.org/awards/distinguishedPaperAward.html

contributions across 38 papers, which are listed separately at the end of the references for this paper.

The principal steps of this study are outlined in the right part of Fig. 3 and elaborated below. Each paper in the cohort of the *ICSE distinguished papers* from 2014–2018 was randomly assigned to two reviewers (among the authors of this paper). As the work was not about judging papers, but about understanding, we did not care for any conflicts of interest. The two reviewers independently *extracted information* from the papers to answer the set of design science questions listed in Table 1. This set of questions were derived from the constituents of our conceptualization of design science as a paradigm.

The first author derived an initial list of questions, which were reviewed by the other authors with minor changes. Since the visual abstract also represents the constituents of design science, the questions map to each of the elements in the visual abstract template. Thus, we *defined a visual abstract* for each paper, which we iterated until we arrived at an agreement for a shared response to these questions, seeking additional input through *peer reviews* by the rest of the research groups of the authors or other experts for papers on topics unfamiliar to us.

The answers to the questions were captured in a spreadsheet to facilitate future analysis as well as ongoing review and internal auditing of our process. Our combined responses were then used to populate the visual abstract template for each paper, from which we analyzed the *design knowledge* contributions in SE (RQ 2 and 3). The collection of visual abstracts for all of the papers is available online at dsse.org, which constitutes our combined understanding of the analyzed software engineering research from a design science perspective.

As part of our analysis, we wanted to validate our interpretations with the *original authors*. To assess the value of such validation, we confirmed our interpretations of the 2014 and 2015 ICSE distinguished papers with the original authors. We heard back from half of the authors of this set of papers, who confirmed the accuracy of our responses (mentioning minor improvements only). We assessed this feedback as a preliminary validation of our process and did not see the need to repeat this step for the other papers—in each case, the abstracts for all papers we studied are available online[4] and the authors may comment publicly on our interpretations if they choose.

Once we finished creating all the visual abstracts, we *clustered* the papers (see the rightmost part of Fig. 3) to identify *contribution types* (RQ1). As we answered the questions in Table 1, we presented our answers to other members in this paper's author team for *peer review*, which in many cases led to refinements in our responses. We also printed the visual abstracts we created for each paper (in miniature), and working as a group in a face-to-face meeting, we sorted the visual abstracts into clusters to identify different types of design science contributions.

Following this face-to-face visual abstract clustering activity, we worked again in pairs to inspect each of the papers in the clusters to confirm our categorization. Then, we reviewed and confirmed the categorization of each paper as a group. During this confirmation process, we refined our categorization and collapsed two categories into one: we combined papers that were initially classified as exploratory with papers that we initially thought were design science contributions in terms of problem understanding, but on reflection these were better framed as explanatory papers as the investigated problems were not linked to a specific

[4] dsse.org

Table 1 Characterizing research through a design science lens: The answers to the following questions were used to populate a visual abstract for each paper

1.	*Problem instance*
1.1	What problem is addressed in the paper? (Describe in terms of the concrete instance of the problem studied.)
2.	*Problem understanding approach*
2.1	How did the authors gain an understanding of the problem?
3.	*Proposed solution(s)*
3.1	What intervention(s) was proposed to solve the identified problem?
4.	*Design approach*
4.1	How did the authors arrive at their proposed solution?
5.	*Validation approach*
5.1	How did the authors apply the intervention/solution to the problem instance to validate it?
6.	*The Technological Rule*
6.1	What effect do they wish to achieve through their research?
6.2	In what situations does this rule apply?
6.3	In summary, what is the proposed solution in the paper?
7.	*Relevance, convincing the target stakeholder*
7.1	What class of problems and solutions are captured by the technological rule?
7.2	To whom are those problem-solution pairs relevant?
7.3	How do the authors convince their readers that the problem-solution pair is relevant to those stakeholders?
8.	*Rigor*
8.1	What actions have been taken to ensure the understanding of the problem instance is valid?
8.2	What actions have been followed to ensure the intervention is a valid solution to the problem instance?
8.3	What actions have been taken to validate the design choices?
9.	*Novelty*
9.1	What are the novel contributions in the paper?

solution. We present the stable clusters that emerged from these activities in the following section of this paper.

4 Results from the Paper Cluster Analysis

Overall we identified five clusters, described in detail below, based on our analysis of how each paper contributed to the extracted technological rule. Note the rules are not extracted by the original authors of the papers but by us for the purpose of this particular review of the papers according to their design science contributions.

1. *Problem-solution pair*: this cluster represents papers that balance their focus on a problem instance and solution.
2. *Solution validation*: this cluster is characterized by papers that concentrate largely on the solution and its validation, rather than on problem conceptualization.
3. *Solution design*: papers in this cluster focus on the design of the solution more than on problem conceptualization or solution validation.
4. *Descriptive*: these papers address a general software engineering phenomenon rather than a specific instance of a problem-solution pair.
5. *Meta*: this cluster of papers may be any of the types above but are aimed at contributing research insights for researchers rather than for practitioners.

Figure 4 illustrates how the first four clusters (1–4) map to a design science view, including both the problem-solution dimension and the theory-practice one. Clusters 1–3 each represent different types of design science research contributions since the papers in these clusters consider explicit problem-solution pairs. Papers in the 4th cluster contribute explanatory knowledge and, although such knowledge may support software engineering solution design, they are better framed through an explanatory lens. Cluster 5 is not represented in this figure as these papers produce knowledge for software engineering researchers rather than for software engineering practitioners.

Figure 5 presents a visual representation of the main clusters that emerged from our analysis along with a listing of which papers (first author/year) belong to the different clusters. The two axes of this graph are defined as follows: the x-axis captures the solution contribution ranging from high-level recommendations to more concrete solutions that are designed and may be validated; and the y-axis indicates the problem understanding contribution, whereby the problem is already known or assumed, to where new insights are produced from the research.

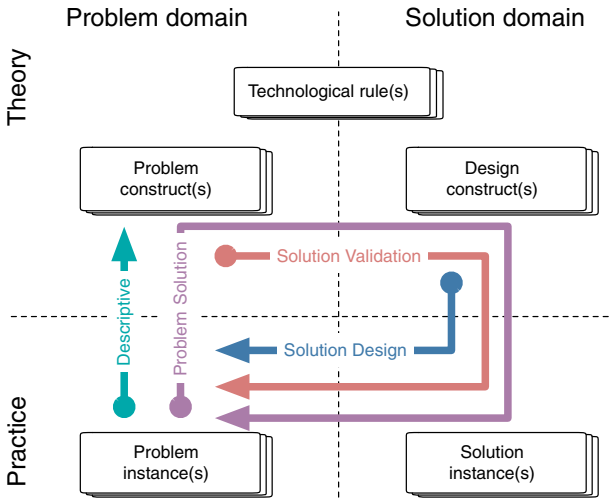

Fig. 4 An illustration of how the identified clusters map to the problem/solution and the practice/theory axes respectively. The arrows show how typical studies in each cluster traverse the four quadrants (1. practical problem, 2. conceptual problem description, 3. general solution design, and 4. instantiated solution)

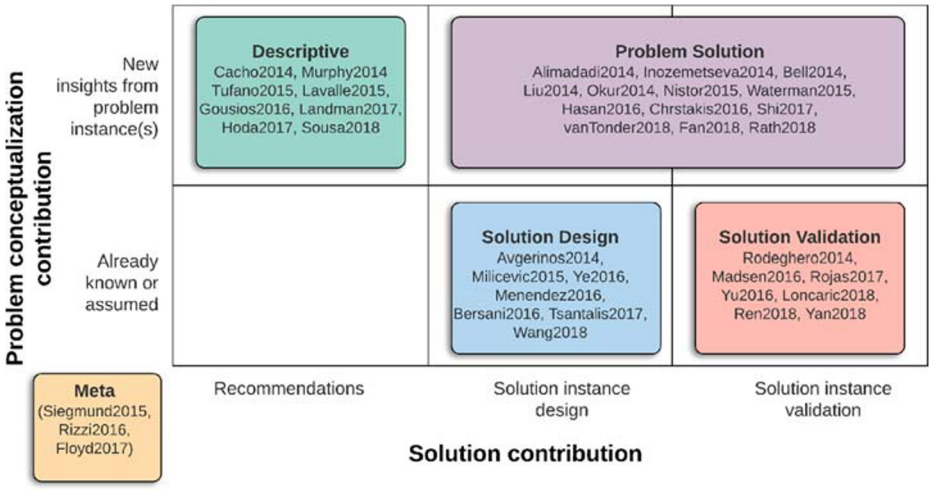

Fig. 5 The main clusters that emerged from our analysis of the papers, showing the key design science contributions in terms of problem understanding insights and solution recommendations, design and/or validation

A more detailed and nuanced description for each cluster is provided below. For each cluster we refer to examples of papers from our analysis and include one visual abstract for each example to showcase the design knowledge that is or is not captured by each cluster.

4.1 Problem-Solution Pair

For the papers in this cluster, a problem instance is identified and investigated to gain a generalized problem formulation matching the proposed solution. A solution is proposed, designed and implemented, then validated rigorously in-context through empirical methods. It is the most populated cluster, indicating that many software engineering papers can be framed in accordance with a design science paradigm.

The technological rule is defined quite clearly in all of the papers belonging to this cluster and is in most cases a new proposal of either a tool or methodological approach necessary to solve the problem instance (see Fig. 6). Consequently, the relation between problem (e.g., difficulty of estimating energy consumption in Java Collection classes) and solution (e.g., use per-method energy profiles to chose among Java Collections and reduce/optimize energy consumption) is explicit.

Solutions are geared towards both practitioners and researchers, making it explicit and easy for a stakeholder to assess the relevance of the rule for their specific case. The solutions are mainly validated by conducting case studies on real projects (Nistor et al. 2015) or controlled experiments (Alimadadi et al. 2014; Bell and Kaiser 2014).

In some cases, alternative solutions are compared to the proposals made. For example, Rath et al. (2018) considered alternative information retrieval techniques and classifiers during the design of their solution, and used precision/recall values collected from all of the compared solutions to develop their classifier.

A representative example for this cluster is the paper by Hasan et al. (2016). The problem instance outlines how it is difficult to estimate energy consumption in Java Collections

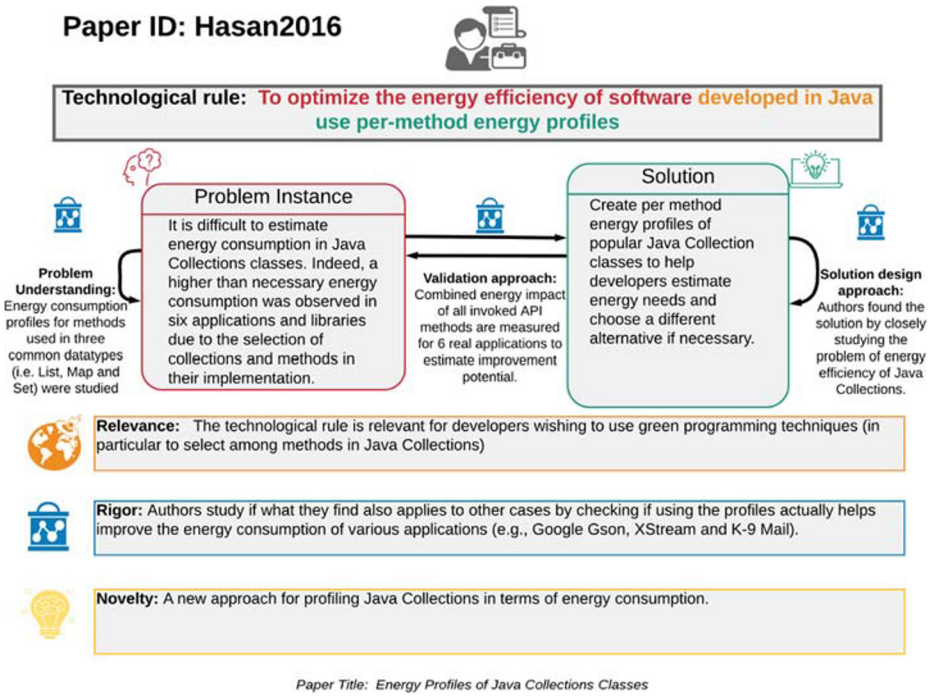

Fig. 6 Visual abstract of a typical paper in the problem solution cluster, Hasan et al. (2016)

classes. As a solution, the authors created detailed profiles of commonly used API methods for three Java Collections datatypes, namely: List, Map, and Set, and validated them through a case study on Java libraries and applications. In this way, developers can use the information of the usage context of a data structure and the measured energy profiles to decide between alternative collection implementations and optimize their solutions. The visual abstract is shown in Fig. 6. Other visual abstracts in this cluster (and other clusters) are available on our online website.[5]

In summary, the problem-solution cluster papers can be seen as presenting complete design science contributions, considering both the general and specific aspects of a problem-solution pair investigated in context, with implications for researchers and practitioners.

4.2 Solution Validation

Papers in the solution validation cluster mainly focus on refining a previously proposed, but often implicit, technological rule. The problem is implicitly derived from a previous solution and its limitations rather than from an observed problem instance. Accordingly, in most cases, the problem is motivated by a general statement at an abstract level, making claims about "many bugs..." or "it is hard to...". Some of the papers underpin these claims with references to empirical studies, either the authors' own studies or from the literature, while others ground the motivation in what is assumed to be generally "known".

[5] dsse.org

As a typical example for this cluster, Loncaric et al. (2018) identify that others have tried to automate the synthesis of data structures and present a tool that embeds a new technique that overcomes the limitations of previous work. A proof of concept is demonstrated through four real cases. The corresponding visual abstract is presented in Fig. 7.

Note that some of the papers in this cluster focus on understanding the problem with previous solutions, with the aim to improve the solution or come up with a new solution. For example, Rodeghero et al. (2014) attempt to improve code summarization techniques for program comprehension. They perform an extensive eye-tracking study to design a code summarization tool.

The technological rules are mostly implicit in these papers. As they are related to problems with existing solutions, rather than original problems in the SE domain, the presentation of the solutions are mostly related to previous solutions. A technological rule can sometimes be derived indirectly, through the aim of an earlier solution, but it is rarely defined explicitly.

The papers in this cluster discuss relevance to research explicitly, while the relevance to practice is mostly discussed indirectly, and at a high abstraction level. For example, Rojas et al. (2017) claim that writing good test cases and generating mutations is hard and boring, and thus they propose a gaming approach to make this more enjoyable and better. The validation is conducted, testing a specific code instance, while the original problem is rooted in high-level common sense knowledge. However, there are other papers in the cluster that back up the problem through evidence, such as a vulnerability database, used by Yan et al. (2018) to motivate addressing the vulnerability problem of Use-After-Free pointers.

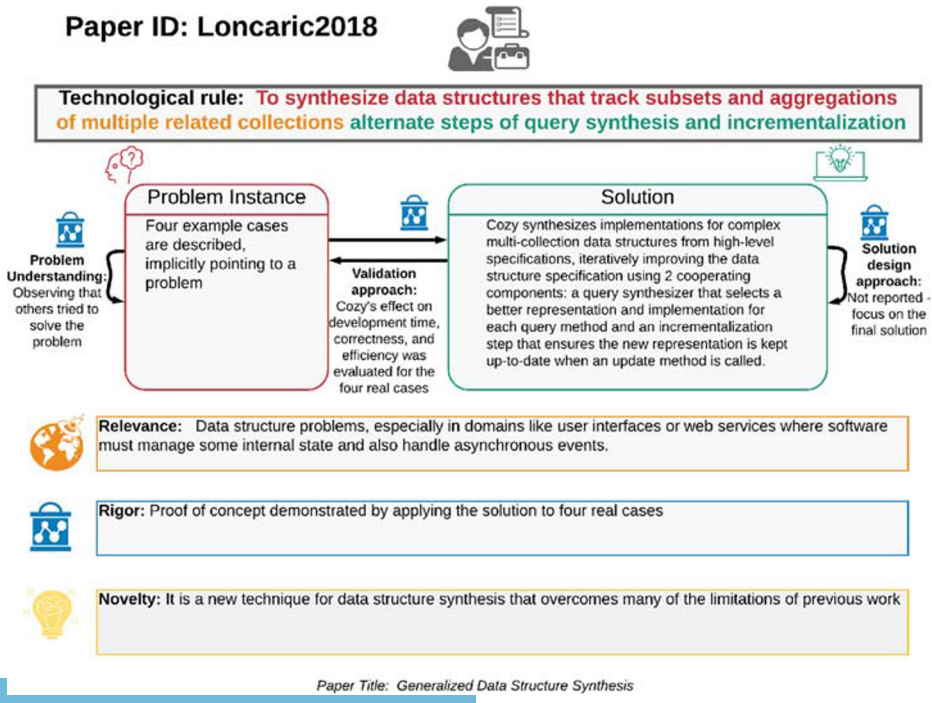

Fig. 7 Visual abstract of a typical paper in the cluster of solution validation studies, Loncaric et al. (2018)

In summary, the solution validation papers focus on refining an existing technological rule. The motivating problem is mostly expressed in terms of high-level knowledge, rather than specific instances, although some papers refer to empirical evidence for the existence and relevance of the problem. The more specific problem description is often related to problems with previous solutions. The papers clearly show a design science character, although they are at risk of solving academic problems, rather than practitioners' problem instances.

4.3 Solution Design

The papers in this cluster present details of a new instantiation of a general solution. For example, Avgerinos et al. (2014) present a new way of testing with symbolic execution, see Fig. 8. The presented approach finds more bugs than the previously available methods. However, the need for this tool was not explicitly stated and the authors perhaps assume the need is clear.

Similarly, Bersani et al. (2016) propose a new semantics for metric temporal logic (MTL) called Lazy Semantics for addressing memory scalability. The proposal builds on previous research and is focused on the solution, a new trace checking algorithm. A similar observation can be made for analysis and validation. For example, the analysis in Avgerinos et al. (2014) is conducted by using the proposed solution on a rather large code base and using well known metrics such as number of faults found, node coverage, and path coverage. Whereas in Bersani et al. (2016), the validation is carried out comparing the designed solution with other, point-based semantics.

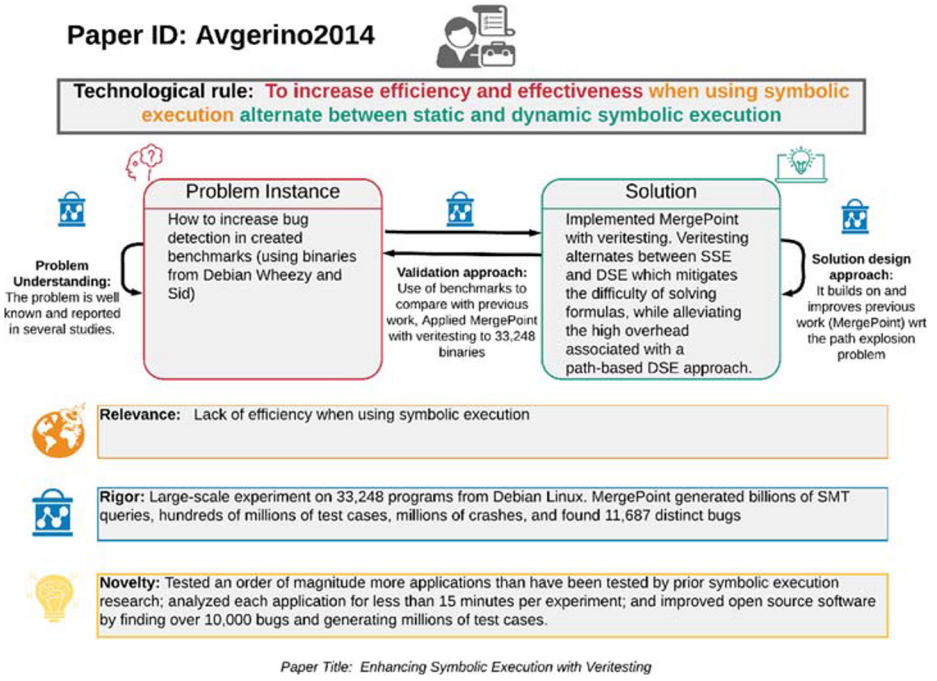

Fig. 8 Visual abstract of a typical paper in the solution design cluster, Avgerinos et al. (2014)

For papers in this cluster, the problem is not explicitly formulated but it is more generally discussed in terms of, for example, decreasing the number of faults. The papers tend to describe the designed solutions in rather technical terms. This is also how the novelty typically is highlighted. Validations are often conducted by applying the proposed solution on a code base and analyzing metrics of, e.g., the number of faults found in testing, and no humans are directly involved as subjects in validations. Empirical data for the validations are either obtained by technically measuring e.g., execution time, or by using data already published in programmer-forums.

In summary, the solution design papers focus on low level technological rules. The motivating problem is in most cases technical details of a solution to a more general problem. While the validity of the general solution is implicit, the low level solution is often validated through controlled experiments or benchmarking in a laboratory setting. The papers clearly show a design science character, although at a low abstraction level.

4.4 Descriptive

The papers categorized in this cluster develop an understanding of a software engineering phenomenon that is currently not well understood. Such research studies may expose problems that need to be addressed, or they may reveal practices or tools that could benefit other challenging software engineering scenarios.

For example, Murphy-Hill et al. (2014) conducted a study of game developers and identify a number of recommendations for how game developers could be better supported through improved tools or practices, while Hoda and Noble (2017) carried out a grounded theory study to achieve an understanding of how teams transition to agile.

Concrete instances of software engineering phenomena have been studied in various ways. Gousios et al. (2016) surveyed 4,000 open source contributors to understand the pull-based code contribution process, Tufano et al. (2015) analyzed git commits from 200 open source repositories to investigate more about code smells, Cacho et al. (2014) studied changes to 119 versions of code extracted from 16 different projects to understand trade-offs between robustness and maintenance and Lavallée and Robillard (2015) reported on a 10 month observational study of one software development team to understand why "good developers write bad code".

Figure 9 shows a typical example of a visual abstract from this cluster. The theoretical contributions of these studies are descriptive problem characterizations. In four out of eight papers, a list of recommendations is provided as well. Thus, it is in most cases possible to derive several technological rules from each such paper. However, these technological rules are not instantiated or evaluated further, and neither are they highlighted as the main contributions of the reported studies.

All papers in this cluster discuss relevance to practice: many explicitly discuss how common the phenomenon under study is (e.g., Gousios et al. 2016 show a diagram of the monthly growth of pull request usage on GitHub). Others implicitly highlight a knowledge gap assumed to be of importance (e.g., Lavallee et al. 2015 pinpoint the lack of knowledge about the impact of organizational factors on software quality). Novelty or positioning is, however, not described in terms of the problem or the solution but about aspects of the study as a whole. Gousios et al. (2016) add a *novel perspective*, the contributors' code review, Lavallée and Robillard (2015) add *more empirical data* about organizational factors and software quality, and Tufano et al. (2015) claim to report the *first empirical investigation* of how code smells evolve over time.

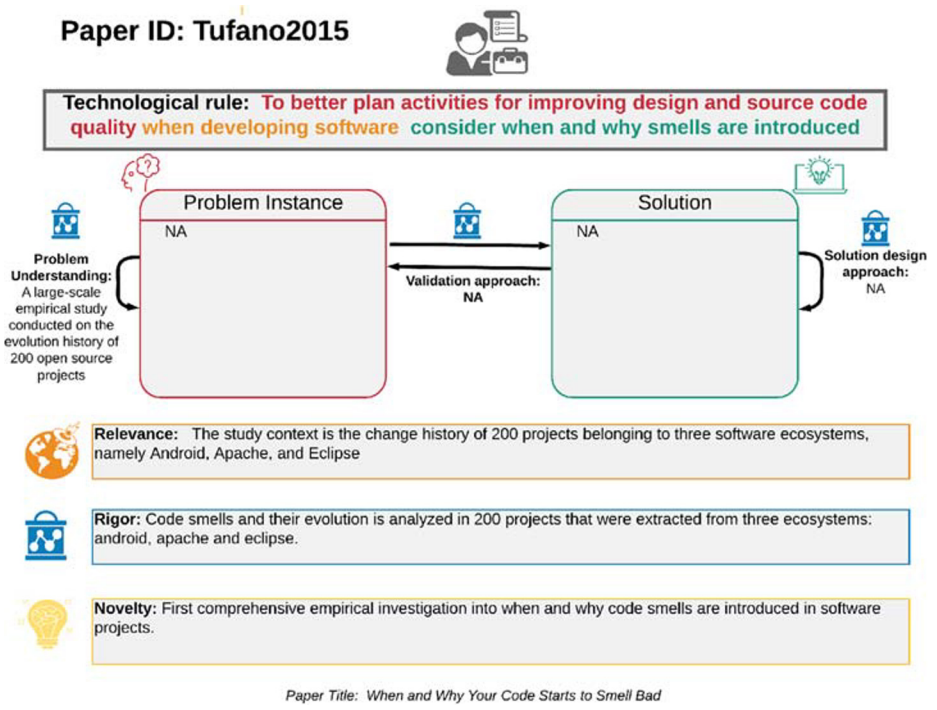

Fig. 9 Visual abstract of a typical paper in the cluster of descriptive studies, Tufano et al. (2015). In this case the visual abstract template does not match the type of study, why some boxes are left empty (NA)

In summary, although the descriptive papers may contribute to design knowledge, i.e., understanding of conceptual problems and initial recommendations, design knowledge in the form of technological rules are not directly described in the papers. The main contributions are discussed in more general terms such as descriptions of the phenomenon under study (defined in the titles) and general information about the study approach and the studied instances (which often appears in the abstracts). Potential problems and their solutions are described in the discussion sections of the papers. Their relevance to practice is in terms of the real-world problems or recommendations that other applications have that tend to be exposed by these kind of papers. Thus, such papers are typically reporting on exploratory research that may be quite high in novelty.

4.5 Meta

Three of the distinguished papers we reviewed do not aim to identify or solve software engineering problems in the real world. Rather, these studies aim at identifying or solving problems which software engineering *researchers* may experience. We therefore refer to them as Meta studies, i.e., addressing the meta level of software engineering *research* in contrast to the primary level of software engineering *practice*. Siegmund et al. (2015) conducted a study that reveals how the software engineering research community lacks a consensus on internal and external validity. Rizzi et al. (2016) advise researchers about how to improve the efficiency of tools that support large-scale trace checking. Finally, Floyd et al. (2017)

propose how fMRI methods can help software engineering researchers gain more insights on how developers comprehend code, and in turn may improve comprehension activities. We show the visual abstract for the Floyd et al. (2017) paper in Fig. 10.

Meta papers address software engineering *research* problems, and propose solutions for software engineering researchers. The design knowledge gained in these studies is primarily about the design of software engineering *research*, and the key stakeholders of the technological rule are rather the *researchers* rather than software engineers. Still, they fall under a design science paradigm and the Meta category of papers may show relevance to industry but in an indirect manner.

In summary, papers that we describe as Meta may fall under a design science research paradigm, leading to a technological rule with *researchers* rather than software engineers as the key stakeholders.

5 Discussion: Design Science Contributions in Software Engineering

The long term goal of much software engineering research is to provide useful recommendations on how to address real-world problems providing evidence for benefits and potential weaknesses of those recommendations. Our analysis of ICSE distinguished papers reveals empirical contributions (RQ1) related to *problem conceptualization*, *solution design*, *solution instantiation*, and *empirical validation* (see the path traversal in Fig. 4). In 13 of the 38 papers we analyzed, all four activities are explored in equal depth while others focus on

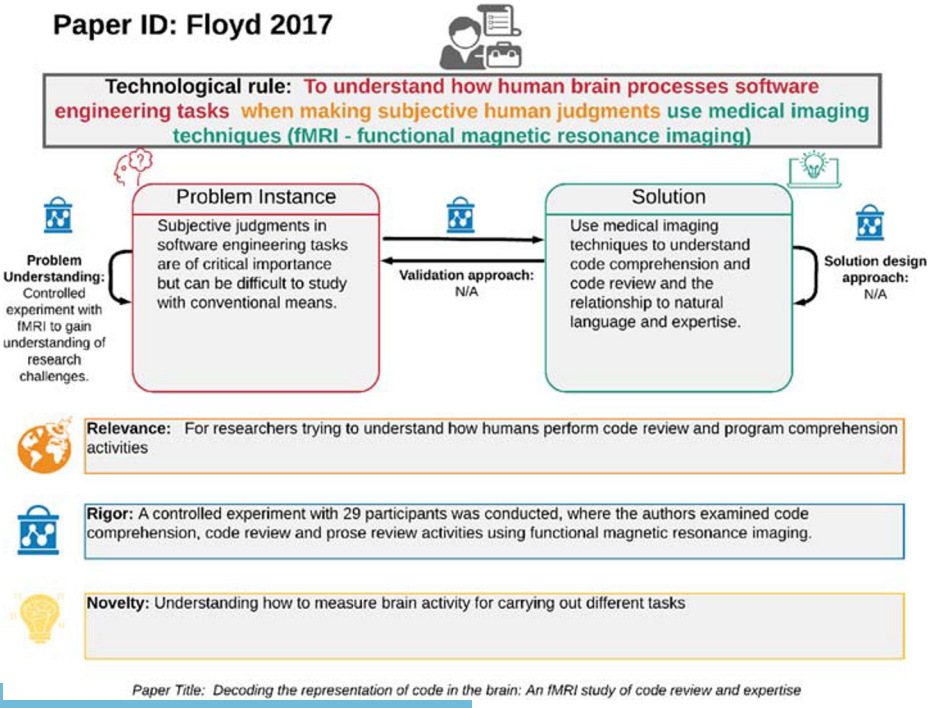

Fig. 10 A typical example of a visual abstract in the Meta cluster, Floyd et al. (2017)

one or two activities, as shown in the clusterings above in Section 4. All of those activities generate knowledge corresponding to the elements of our visual abstract template (see Fig. 2). However, none of the papers are *presented* in terms of these elements and we had to spend significant effort using the questions in Table 1 to extract this knowledge in a systematic way. Extracting technological rules from most papers was also quite challenging. That said, applying the design science lens helped us notice and distinguish the different kinds of design contributions from the papers we analyzed, and guided our assessment of the papers in terms of research relevance, rigor and novelty. We discuss our experiences using the design science lens below and also the challenges we faced applying it to different types of papers. We also allude to experiences we have faced as researchers and reviewers of research papers below.

5.1 Problem Conceptualization and Descriptive Research in Software Engineering

We found a design science paradigm helped us distinguish descriptive research contributions from prescriptive research contributions in the papers we analyzed. Indeed eight of the papers we analyzed focused primarily on the understanding of software engineering problems or phenomenon that were not currently well understood. Descriptive papers are often labeled by the authors as "exploratory" research. Often these papers do not only describe or expose specific problems or phenomenon, but they may also describe why or how certain solutions or interventions are used, and conclude with directions for future research or with recommendations for practitioners to consider (e.g., to use a studied intervention in a new context).

We struggled at first to apply the design science lens to some of these descriptive papers as for most of them no explicit intervention or recommendations were described or given. Articulating clear technological rules was not possible as this research does not aim at producing design prescriptions (yet). However, on reflection we recognized that the design science lens helped us to recognize and appreciate the longer term goals behind this exploratory research that would later culminate in design knowledge. Sometimes we found that descriptive research is under appreciated over prescriptive solutions, but understanding problems clearly is also an important research contribution in a field like software engineering that changes rapidly. In fact, often researchers are "catching up" to what is happening in industry and to recognize new emerging problems that may arise in industrial settings as tools and practices evolve.

Another cluster of papers we identified, the 13 problem-solution pair papers, also contribute insights on problems experienced in software engineering projects. Many of the problem-solution papers derive problem insights from specific problem instances. This was the biggest cluster of papers. The design science lens helped us recognize and appreciate that these papers had contributions, not just on the solution design and validation side, but also contributed or confirmed insights on a studied problem. We have all had experiences when reviewing papers where a co-reviewer failed to recognize problem understanding contributions and argued that a given solution was either too trivial or poorly evaluated. As papers are written (and then typically read) in a linear fashion, losing track of the various contributions can happen. For us, laying out the contributions visually (and by answering the questions we explicitly posed in Table 1) helped us keep track of and appreciate contributions on both the problem and solution aspects.

5.2 Solution Design Contributions in Software Engineering Research

The other two main clusters of papers that are aimed at improving software engineering practice are the seven solution-design papers and seven solution-validation papers. These papers contribute *design knowledge* concerning an intervention and mostly rely on either previous research or accepted wisdom that the problem they address is in need of solving. For these papers, the first questions in Table 1 about the problem instance addressed and problem understanding approach did not always have an explicit answer in the paper. However, to conduct an empirical validation of the design, some kind of instantiation of the problem is required and we referred to these instances when extracting information about problem instance and problem understanding for our analysis. We found this to be an effective way to address distances between the abstraction level of the proposed technological rule and its empirical validation. Papers without specified problem instances are at risk of proposing solutions, which do not respond to real software engineering problems.

5.3 Identifying Technological Rules from Software Engineering Research

For most papers, we were able to extract *technological rules* from the presented research. However, none of the papers had any conclusion or recommendation in such a condensed form (see RQ2). In some cases, the abstracts and introduction sections were written clearly enough that we could identify the intended effect, the situation and the proposed solution intervention presented in the paper. Moreover, when research goals and questions were explicitly stated, technological rules were easier to formulate. Other papers required more detailed reading to extract the needed information. In some publication venues, structured abstracts are introduced as a means to achieve similar clarity and standardization (Budgen et al. 2008), but such abstracts are not typically used for ICSE. Introducing technological rules would, we believe, help in communicating the core of the contribution, both to peer academics and potentially also to industry. Development towards more explicit theory building in software engineering (Sjøberg et al. 2008; Stol and Fitzgerald 2015) may also pave the way for technological rules as a means to express theoretical contributions.

5.4 Assessing Design Knowledge Contributions: Rigor, Relevance and Novelty

Our analysis of rigor, relevance and novelty are based on questions 7-9 in Table 1. *Rigor* can be considered in terms of the suitability of specific research methods used or in how a certain method is applied. Empirical research methods – quantitative as well as qualitative – fit well into both problem conceptualization and validation and we saw examples of very different methods being used. How rigor is ensured of course depends on the choice of method as we discuss above. We found that most authors discussed *rigor*—not surprising given that these papers were considered as the best papers from an already competitive publishing venue (see RQ3). Whether rigor was discussed for the steps of problem conceptualization, solution design and empirical validation, depended on the paper cluster. The *solution validation* and *solution design* papers tended to rigorously benchmark their solution against a code base or other artifacts to demonstrate the merits of the proposed approach. We found that validating the solutions in industrial contexts was not common in these two clusters of papers. Consequently we also found that *relevance* in terms of specific stakeholders was not

discussed much in these papers (as compared to the *descriptive* or *problem-solution* clusters of papers).

How *novelty* was discussed by authors varied greatly depending on the paper cluster but also by the author. As the papers did not explicate the technological rules, none of them discussed their contribution in terms of technological rule novelty. Descriptive papers tended to focus on novelty of the described problem or phenomenon, problem-solution and solution-design papers focused on novelty of the solution, and solution-validation papers emphasized the solution (if refined or new) and insights from the solution validation.

6 Recommendations for Software Engineering Research

As researchers (and reviewers) ourselves we find that contributions from research papers are often not evident, and thus interested researchers and reviewers may miss the value in the papers. Furthermore, the technological rules, even for papers that aim at producing these rules, are not always easy to identify. To help in the design of research and perhaps also in the review of papers, we suggest using the design lens as follows:

- *Explicate design science constructs*: We found design science constructs in most papers, but presenting each of the constructs explicitly, e.g., through the visual abstract (Storey et al. 2017), could help in communicating the research contributions to peer researchers and reviewers. Expressing the technological rules clearly and at a carefully selected level of abstraction, help in communicating the *novelty* of the contributions and may help in advancing the research in "standing on each others shoulders".
- *Use real problem instances*: Anchoring research in real problem instances could help to ensure the *relevance* of a solution. Without reference to an explicit problem instance, the research is at risk of losing the connection with the original question, as the details of a particular intervention are described or assessed by others.
- *Choose validation methods and context*: *Rigor* in terms of method choice is an important consideration. The choice of methods and context for the validation may be different, depending on the intended scope of the theoretical contribution (i.e., the technological rule). If the scope is focused on fine tuning the design of an intervention, stakeholders may not need to be directly involved in the validation. If however the scope includes the perspective of stakeholders and their context, then methods and study contexts should reflect these perspectives.
- *Use the design science lens as a research guide*: The visual abstract and its design science perspective may also be used to guide the design of studies and research programs, i.e., setting particular studies in a particular context. Similarly, a design science perspective can be used as an analysis tool in mapping studies (Petersen et al. 2008), to assess existing research and identify research gaps that can be explored in future research studies and lead to *novel* contributions.
- *Consider research design as a design science*: The cluster of *meta* studies, which are primarily aimed for researchers as the stakeholders, indicate that the design science lens also fits for the design and conduct of studies that focus on understanding our research methodology and methods. Papers that address problems in conducting research and propose solutions to help achieve higher quality research contributions are important contributions for our community to reflect and grow in research maturity. Conducting and presenting these in the same way as studies in the software engineering domain adds to their credibility and emphasizes how they are *relevant* to our community. This

paper is also an example of a meta study aimed at our research community members as stakeholders. We created a visual abstract for this paper as well, and it may be found with our online materials (at dsse.org).

We hypothesize that following these recommendations, based on our in depth analysis of ICSE distinguished papers, would enable a more consistent assessment of rigor, relevance and novelty of the research contributions, and thus also help the peer review process for future conference and journal publications.

7 Limitations

In order to understand how design science can be a useful lens for describing software engineering research, we considered all papers that have received a distinguished paper award over a five year period within a major venue such as ICSE. We felt these may represent papers that our community considers relevant and fine exemplars of SE research. We acknowledge that we would likely see a different result for a different population of papers (e.g., all papers presented at ICSE or in other tracks or venues or journals). That said, we purposefully selected this sample of papers as an exploratory step.

Our view of design science may differ from other views that are reported in the literature. We developed it from examining several interpretations of design science as discussed in Storey et al. (2017) and in Section 2. Our common view of design science in software engineering was developed over the course of two years spent reading and discussing many design science papers in related domains. Our interpretation was developed in an iterative manner. We have used our visual abstract template in several workshops (notably at ISERN 2017,[6] RET 2017[7] and CBSoft 2019[8]) and received favorable feedback about the viable application of the template to software engineering papers that contain design knowledge. However, we recognize that applying the visual abstract to papers is not always straightforward and we found that even among our team, we would apply it differently and pull out different highlights from the papers we read. We found that the process of applying it is in the end more important than the actual text we put in the boxes as doing so helped us understand the main contributions in the papers we analyzed from a design science perspective.

We recognize that our interpretations of the research contributions from the papers we examined may not be entirely accurate or complete. For this reason we requested feedback from the authors of the papers from 2014 and 2015 to check that our view of the design knowledge in their papers was accurate based on our understanding of their work. Among the responses we received (7 of 14 of the paper authors responded), all but one agreed with our summaries presented through the visual abstracts, while the sole initial disagreement was due to misinterpretation of the visual abstract template. This feedback convinced us that we were proceeding in the right direction. Consequently, we decided to rely on our judgment alone for the remaining papers.

To do our own validation, we divided papers equally among us assigning two of us to each paper. We would independently answer the design science questions (as mentioned in Section 3), then refer back to the paper in cases of disagreement, and merge our responses

[6] http://www.scs.ryerson.ca/eseiw2017/ISERN/index.html
[7] https://dl.acm.org/citation.cfm?id=3149485.3149522
[8] https://github.com/margaretstorey/cbsoft2019tutorial

until we reached full agreement. With cases of ongoing disagreement, we sought additional expert opinions. Finally, we reviewed all of the abstracts as a group to reconfirm our interpretations. These abstracts are available online and open for external audit by the paper authors or by others in the community.

To derive clusters of the papers, we followed a rigorous process. We met face to face in a several hour workshop and followed up in several sessions to derive the clusters and categorize and reconfirm the categorization of the papers. We recognize that how we clustered the papers is potentially subjective and others may feel papers belong in different clusters, and may also find different clusters. We have posted all of the visual abstracts and our cluster diagram online which links to all of the visual abstracts (see https://doi.org/10.5281/zenodo.3465675). We welcome comments on our clusters and the categorization of individual papers.

8 Related Work

In this paper, we introduced our conceptualization of design science and the visual abstract template which instantiates our conceptualization and was designed to support communication and dissemination of design knowledge. Furthermore, we reviewed a cohort of software engineering research papers through this lens to understand the design science contributions in the papers. In this section of the paper, we extend the scope of related work to include other conceptualizations of design science, as well as other reviews of design science research conducted in a related field.

Design science has been conceptualized by Wieringa in software engineering (Wieringa 2009) and by several researchers in other disciplines, such as information systems (Hevner et al. 2004; Gregor and Hevner 2013; Johannesson and Perjons 2014) and organization and management (van Aken 2005). Wieringa describes design science as an act of producing knowledge by designing useful things (Wieringa 2009) and makes a distinction between knowledge problems and practical problems. Similarly, Gregor and Hevner emphasize the dual focus on the artifact and its design (Gregor and Hevner 2013) in information systems, and argue for an iterative design process where evaluation of the artifact provides feedback to improve both the design process and the artifact.

In our paper, we do not distinguish between knowledge problems and solution problems within the design sciences, but stress that the researcher's task is always to produce knowledge, which in turn can be used by practitioners for solving their problems. Such knowledge may be embedded in artifacts such as tools, models and techniques or distilled to simple technological rules. In accordance with van Aken (2005), we distinguish between the explanatory sciences and the design sciences as two different paradigms producing different types of theory (explanatory and prescriptive, respectively) with different validity criteria. This is similar to Wieringa's distinction between knowledge problems and practical problems (Wieringa 2009). In our study, we identified one cluster of software engineering papers belonging to the explanatory sciences (descriptive) and three clusters of papers belonging to the design sciences (problem-solution, solution-design, and solution evaluation)

In the management domain, van Aken propose to distinguish management theory, that is prescriptive, from organizational theory, that is explanatory (van Aken 2005). A corresponding division of software engineering theory has not been proposed yet, although theory types are discussed by software engineering researchers (Sjøberg et al. 2008; Stol and Fitzgerald 2013).

In the literature, design science has been studied and is thereby conceptualized in a number literature studies, which are relevant for this study. In the area of information systems,

several literature reviews were conducted of design science research. Indulska and Recker (2008) analyzed design science articles from 2005–07 from well-known information systems conferences. They identified 142 articles, which they divided into groups, such as methodology- and discussion-oriented papers and papers presenting implementations of the design science approach. They found an increasing number of design science papers over the studied years.

Deng et al. (2017) and Deng and Ji (2018) also published a systematic review of design science articles in information systems. They identified articles by searching in top information systems journals and conferences from the years 2001–15, filtering the results and applying snow-balling, resulting in a final review sample of 119 papers or books. In their review, they analyze the topic addressed, artifact type, and evaluation method used. In our review, we classified papers along another dimension, i.e., what types of software engineering design science contributions the papers present in terms of problem understanding, solution design and solution validation. To our knowledge no reviews of software engineering literature have been made from a design science perspective before.

Wieringa et al. (2011) analyzed reasons for the low use of theories in software engineering by studying a set of papers identified in Hannay et al. (2007). They compare identified theories in software engineering to general theories with respect to level of generalization, form of theory, and use of theory, and argue that the reasons for low use of theories have to do with idealizing assumptions, context of software engineering theories, and that statistical model building needs no theories.

Concerning research relevance, Beecham et al. communicated with a test group of practitioners (Beecham et al. 2014) and found that evidence based on experience was seen as most important, and if it was not available in their own organization, they would seek information from similar organizations in the world for insights on global software engineering. They compare typical sources for software engineering researchers and sources where practitioners seek information and found that the overlap is very small. Similar findings were obtained by Rainer et al. (2003) study based on focus groups with practitioners and a literature review. These observations point to the need for presenting research in a way that is useful for practitioners. This is also discussed by Grigoleit et al. (2015), who conclude that practitioners assess the usefulness of many artifacts as being too low. This is in line with our findings, where we put forward the design science lens as a means to better communicate prescriptive research contributions in software engineering. That said, we have not evaluated it with practitioners so far.

Another attempt to make evidence available to practitioners is presented by Cartaxo et al. (2016). They present the concept of "evidence briefings", which is a way to summarize systematic literature reviews in a one-page format. They used accepted information design principles to design the structure of the one-page briefing. The format and content were positively validated by both practitioners and researchers. While evidence briefings may provide an effective way to synthesize evidence from several studies, our visual abstract template may provide a way to effectively summarize the contribution of one study or research program from a design science perspective.

9 Conclusions and Future Work

Design science, although suggested for some time as a useful research paradigm for software engineering research, is not commonly used as a way to frame software engineering

research contributions. Yet our analysis of 38 ICSE distinguished papers indicates that the majority of these papers can be expressed in terms of a design science paradigm. Much software engineering research is solution oriented, providing design knowledge, although it is less clear which problems some papers aim to solve.

The technological rule, as a condensed summary of design knowledge, offers a means to communicate not just the solutions designed or validated, but also the problems addressed. We were able to derive technological rules from most papers, although they were not explicitly stated as such in these papers. In future work, we aim to investigate how technological rules could be linked across different research contributions that address the same underlying problem. A higher level technological rule could be decomposed into more narrow but related rules, thus bringing insights across multiple papers that are linked by context, intervention type and effect. Currently, we lack the machinery in our community to link papers at this theoretical level and the results in papers remain as silos and are often not even referenced in related work. The technological rule template could help fill this gap and help us to better understand what we know and what we don't know yet about certain problems and challenges in software engineering.

Also as future work, we wish to investigate if the design science visual abstract (or some variant of it) could provide an efficient way to present software engineering research contributions to industry. We published our abstracts from this study online (at dsse.org) but it remains to be seen if industry finds this format useful or not. We expect that extracting technological rules from a set of papers that address a common problem or topic is likely to be of more value to industry (but this was not a goal of this current work). In the meantime, we anticipate that our analysis of ICSE distinguished papers through the design science lens may help our community increase adoption of the design science lens, which we anticipate in turn will allow us to do a better job of communicating, understanding and building on each others' work.

Furthermore, as a means for spreading the word of this research to the community, it is our intention to contact editors of important journals as well as program chairs of relevant conferences such as ICSE and promote the adoption of VAs for authors that submit a research paper.

Acknowledgements We would like to thank Cassandra Petrachenko for her careful edits of our paper. Daniela Soares Cruzes, Johan Linåker, Sergio Rico and Eirini Kalliamvakou gave us helpful comments on an earlier draft of this paper. We would also like to thank some of the authors of ICSE distinguished papers for giving us feedback, as well as participants at ISERN 2017, RET 2017, and CBSoft 2019 for trying out the visual abstract. Marco Gerosa also gave us valuable insights on the visual abstract and how we applied it. We thank the anonymous EMSE reviewers for helping us sharpening the contribution of this paper. The research was partially funded by the Faculty of Engineering at Lund University through the Lise Meitner guest professorship (Storey), the ELLIIT strategic research area (Engström), and the EASE industrial excellence center (Runeson).

Funding Information Open access funding provided by Lund University.

References

Beecham S, O'Leary P, Baker S, Richardson I, Noll J (2014) Making software engineering research relevant. Computer 47(4):80–83. https://doi.org/10.1109/MC.2014.92

Budgen D, Kitchenham BA, Charters SM, Turner M, Brereton P, Linkman SG (2008) Presenting software engineering results using structured abstracts: a randomised experiment. Empir Softw Eng 13(4):435–468. https://doi.org/10.1007/s10664-008-9075-7

Cartaxo B, Pinto G, Vieira E, Soares S (2016) Evidence briefings: Towards a medium to transfer knowledge from systematic reviews to practitioners. In: Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '16, pp 57:1–57:10

Carver JC, Juristo N, Baldassarre MT, Vegas S (2014) Replications of software engineering experiments. Empir Softw Eng 19(2):267–276. https://doi.org/10.1007/s10664-013-9290-8

Deng Q, Ji S (2018) A review of design science research in information systems: Concept, process, outcome, and evaluation. Pacific Asia journal of the association for information systems, vol 10

Deng Q, Wang Y, Ji S (2017) Design science research in information systems: A systematic literature review 2001-2015. In: CONF-IRM 2017 Proceedings

Dybå T, Sjøberg D, Cruzes DS (2012) What works for whom, where, when, and why? On the role of context in empirical software engineering. In: Proceedings of the 2012 ACM-IEEE international symposium on empirical software engineering and measurement, pp 19–28. https://doi.org/10.1145/2372251.2372256

Gregor S, Hevner AR (2013) Positioning and presenting design science research for maximum impact. MIS Q 37(2):337–356

Grigoleit F, Vetro A, Diebold P, Fernandez DM, Bohm W (2015) In quest for proper mediums for technology transfer in software engineering. In: 2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), pp 1–4. https://doi.org/10.1109/ESEM.2015.7321203

Hannay JE, Sjöberg DIK, Dybå T (2007) A systematic review of theory use in software engineering experiments. IEEE Trans Softw Eng 33(2):87–107

Hevner AR, March ST, Park J, Ram S (2004) Design science in information systems research. MIS Q 28(1):75–105

Indulska M, Recker JC (2008) Design science in IS research : a literature analysis. In: Gregor S, Ho S (eds) 4th Biennial ANU workshop on information systems foundations. ANU E Press, Canberra

Johannesson P, Perjons E (2014) An introduction to design science. Springer Publishing Company, Incorporated

Juristo N, Gómez OS (2010) Replication of software engineering experiments. In: Empirical software engineering and verification. Springer, pp 60–88

Petersen K, Feldt R, Mujtaba S, Mattsson M (2008) Systematic mapping studies in software engineering. In: Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering, EASE'08. BCS Learning & Development Ltd., Swindon, pp 68–77

Rainer A, Hall T, Baddoo N (2003) Persuading developers to "buy into" software process improvement: a local opinion and empirical evidence. In: International Symposium on Empirical Software Engineering, ISESE, pp 326–335

Sedlmair M, Meyer M, Munzner T (2012) Design study methodology: Reflections from the trenches and the stacks. IEEE Trans Vis Comput Graph 18(12):2431–2440. https://doi.org/10.1109/TVCG.2012.213

Sein MK, Henfridsson O, Purao S, Rossi M, Lindgren R (2011) Action design research. MIS Q 35(1):37–56

Shneiderman B (2016) The new ABCs of research: achieving breakthrough collaborations, 1st edn. Oxford University Press, Inc., New York

Shull FJ, Carver JC, Vegas S, Juristo N (2008) The role of replications in empirical software engineering. Empir Softw Eng 13(2):211–218. https://doi.org/10.1007/s10664-008-9060-1

Sjøberg DI, Dybå T, Anda BC, Hannay JE (2008) Building theories in software engineering. In: Guide to advanced empirical software engineering. Springer, pp 312–336

Stol KJ, Fitzgerald B (2013) Uncovering theories in software engineering. In: 2013 2nd SEMAT Workshop on a General Theory of Software Engineering (GTSE), pp 5–14. https://doi.org/10.1109/GTSE.2013.6613863

Stol KJ, Fitzgerald B (2015) Theory-oriented software engineering, vol 101, pp 79–98, https://doi.org/10.1016/j.scico.2014.11.010. Towards general theories of software engineering

Storey MA, Engström E, Höst M, Runeson P, Bjarnason E (2017) Using a visual abstract as a lens for communicating and promoting design science research in software engineering. In: Empirical Software Engineering and Measurement (ESEM), pp 181–186. https://doi.org/10.1109/ESEM.2017.28

van Aken JE (2004) Management research based on the paradigm of the design sciences: the quest for field-tested and grounded technological rules: paradigm of the design sciences. J Manag Stud 41(2):219–246. https://doi.org/10.1111/j.1467-6486.2004.00430.x

van Aken JE (2005) Management research as a design science: articulating the research products of mode 2 knowledge production in management. Br J Manag 16(1):19–36. https://doi.org/10.1111/j.1467-8551.2005.00437.x

Wieringa R (2009) Design science as nested problem solving. In: Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology, DESRIST '09. ACM, New York, pp 8:1–8:12, https://doi.org/10.1145/1555619.1555630

Wieringa R, Daneva M, Condori-Fernandez N (2011) The structure of design theories, and an analysis of their use in software engineering experiments. In: 2011 International symposium on empirical software engineering and measurement, pp 295–304

Wieringa R, Moralı A (2012) Technical action research as a validation method in information systems design science. In: Peffers K, Rothenberger M, Kuechler B (eds) Design science research in information systems. Advances in theory and practice. Springer, Berlin, pp 220–238

Wieringa RJ (2014) Design science methodology for information systems and software engineering. Springer, Berlin. https://doi.org/10.1007/978-3-662-43839-8_1

Wohlin C, Aurum A (2015) Towards a decision-making structure for selecting a research design in empirical software engineering. Empir Softw Eng 20(6):1427–1455. https://doi.org/10.1007/s10664-014-9319-7

References to ICSE distinguished papers

Alimadadi S, Sequeira S, Mesbah A, Pattabiraman K (2014) Understanding javascript event-based interactions. In: Proceedings of the 36th International Conference on Software Engineering, ICSE 2014. ACM, New York, pp 367–377, https://doi.org/10.1145/2568225.2568268

Avgerinos T, Rebert A, Cha SK, Brumley D (2014) Enhancing symbolic execution with veritesting. In: Proceedings of the 36th International Conference on Software Engineering, ICSE 2014. ACM, New York, pp 1083–1094, https://doi.org/10.1145/2568225.2568293

Bell J, Kaiser G (2014) Unit test virtualization with vmvm. In: Proceedings of the 36th International Conference on Software Engineering, ICSE 2014. ACM, New York, pp 550–561, https://doi.org/10.1145/2568225.2568248

Bersani MM, Bianculli D, Ghezzi C, Krstić S, Pietro PS (2016) Efficient large-scale trace checking using mapreduce. In: Proceedings of the 38th International Conference on Software Engineering, ICSE '16. ACM, New York, pp 888–898, https://doi.org/10.1145/2884781.2884832

Cacho N, César T, Filipe T, Soares E, Cassio A, Souza R, Garcia I, Barbosa EA, Garcia A (2014) Trading robustness for maintainability: An empirical study of evolving c# programs. In: Proceedings of the 36th International Conference on Software Engineering, ICSE 2014. ACM, New York, pp 584-595, https://doi.org/10.1145/2568225.2568308

Christakis M, Müller P, Wüstholz V (2016) Guiding dynamic symbolic execution toward unverified program executions. In: Proceedings of the 38th International Conference on Software Engineering, ICSE '16. ACM, New York, pp 144-155, https://doi.org/10.1145/2884781.2884843

Fan L, Su T, Chen S, Meng G, Liu Y, Xu L, Pu G, Su Z (2018) Large-scale analysis of framework-specific exceptions in android apps. In: Proceedings of the 40th International Conference on Software Engineering, ICSE '18. ACM, New York, pp 408-419, https://doi.org/10.1145/3180155.3180222

Floyd B, Santander T, Weimer W (2017) Decoding the representation of code in the brain: An fmri study of code review and expertise. In: Proceedings of the 39th international conference on software engineering. IEEE Press, pp 175–186

Gousios G, Storey MA, Bacchelli A (2016) Work practices and challenges in pull-based development: The contributor's perspective. In: Proceedings of the 38th International Conference on Software Engineering, ICSE '16. ACM, New York, pp 285–296, https://doi.org/10.1145/2884781.2884826

Hasan S, King Z, Hafiz M, Sayagh M, Adams B, Hindle A (2016) Energy profiles of java collections classes. In: Proceedings of the 38th International Conference on Software Engineering, ICSE '16. ACM, New York, pp 225-236, https://doi.org/10.1145/2884781.2884869

Hoda R, Noble J (2017) Becoming agile: a grounded theory of agile transitions in practice. In: Proceedings of the 39th International Conference on Software Engineering. IEEE Press, pp 141–151

Inozemtseva L, Holmes R (2014) Coverage is not strongly correlated with test suite effectiveness. In: Proceedings of the 36th International Conference on Software Engineering, ICSE 2014. ACM, New York, pp 435–445, https://doi.org/10.1145/2568225.2568271

Landman D, Serebrenik A, Vinju JJ (2017) Challenges for static analysis of java reflection: Literature review and empirical study. In: Proceedings of the 39th International Conference on Software Engineering, ICSE '17. IEEE Press, Piscataway, pp 507–518, https://doi.org/10.1109/ICSE.2017.53

Lavallée M, Robillard PN (2015) Why good developers write bad code: An observational case study of the impacts of organizational factors on software quality. In: Proceedings of the 37th International Conference on Software Engineering - Volume 1, ICSE '15. IEEE Press, Piscataway, pp 677–687

Liu Y, Xu C, Cheung SC (2014) Characterizing and detecting performance bugs for smartphone applications. In: Proceedings of the 36th International Conference on Software Engineering, ICSE 2014. ACM, New York, pp 1013-1024, https://doi.org/10.1145/2568225.2568229

Loncaric C, Ernst MD, Torlak E (2018) Generalized data structure synthesis. In: Chaudron M, Crnkovic I, Chechik M, Harman M (eds) Proceedings of the 40th international conference on software engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018. ACM, pp 958–968. https://doi.org/10.1145/3180155.3180211

Madsen M, Tip F, Andreasen E, Sen K, Møller A (2016) Feedback-directed instrumentation for deployed javascript applications. In: Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016, pp 899–910. https://doi.org/10.1145/2884781.2884846

Menendez D, Nagarakatte S (2016) Termination-checking for llvm peephole optimizations. In: Proceedings of the 38th International Conference on Software Engineering, ICSE '16. ACM, New York, pp 191–202, https://doi.org/10.1145/2884781.2884809

Milicevic A, Near JP, Kang E, Jackson D (2015) Alloy*: A general-purpose higher-order relational constraint solver. In: Proceedings of the 37th international conference on software engineering - Volume 1, ICSE '15. IEEE Press, Piscataway, pp 609–619

Murphy-Hill E, Zimmermann T, Nagappan N (2014) Cowboys, ankle sprains, and keepers of quality: How is video game development different from software development? In: Proceedings of the 36th International Conference on Software Engineering, ICSE 2014. ACM, New York, pp 1–11, https://doi.org/10.1145/2568225.2568226

Nistor A, Chang PC, Radoi C, Lu S (2015) Caramel: Detecting and fixing performance problems that have non-intrusive fixes. In: Proceedings of the 37th international conference on software engineering - Volume 1, ICSE '15. IEEE Press, Piscataway, pp 902–912

Okur S, Hartveld DL, Dig D, Deursen AV (2014) A study and toolkit for asynchronous programming in c#. In: Proceedings of the 36th international conference on software engineering, ICSE 2014. ACM, New York, pp 1117–1127, https://doi.org/10.1145/2568225.2568309

Rath M, Rendall J, Guo JLC, Cleland-Huang J, Mäder P. (2018) Traceability in the wild: Automatically augmenting incomplete trace links. In: Proceedings of the 40th international conference on software engineering, ICSE '18. ACM, New York, pp 834–845, https://doi.org/10.1145/3180155.3180207

Ren Z, Jiang H, Xuan J, Yang Z (2018) Automated localization for unreproducible builds. In: Chaudron M, Crnkovic I, Chechik M, Harman M (eds) Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018. ACM, pp 71–81. https://doi.org/10.1145/3180155.3180224

Rizzi EF, Elbaum S, Dwyer MB (2016) On the techniques we create, the tools we build, and their misalignments: a study of klee. In: Proceedings of the 38th international conference on software engineering. ACM, pp 132–143

Rodeghero P, McMillan C, McBurney PW, Bosch N, D'Mello SK (2014) Improving automated source code summarization via an eye-tracking study of programmers. In: 36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014, pp 390–401. https://doi.org/10.1145/2568225.2568247

Rojas JM, White TD, Clegg BS, Fraser G (2017) Code defenders: crowdsourcing effective tests and subtle mutants with a mutation testing game. In: Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017, pp 677–688. https://doi.org/10.1109/ICSE.2017.68

Shi A, Thummalapenta S, Lahiri SK, Bjorner N, Czerwonka J (2017) Optimizing test placement for module-level regression testing. In: Proceedings of the 39th International Conference on Software Engineering, ICSE '17. IEEE Press, Piscataway, pp 689–699, https://doi.org/10.1109/ICSE.2017.69

Siegmund J, Siegmund N, Apel S (2015) Views on internal and external validity in empirical software engineering. In: Proceedings of the 37th international conference on software engineering-Volume 1. IEEE Press, pp 9–19

Sousa L, Oliveira A, Oizumi W, Barbosa S, Garcia A, Lee J, Kalinowski M, de Mello R, Fonseca B, Oliveira R, et al. (2018) Identifying design problems in the source code: a grounded theory. In: Proceedings of the 40th International Conference on Software Engineering. ACM, pp 921–931

van Tonder R, Goues CL (2018) Static automated program repair for heap properties. In: Proceedings of the 40th International Conference on Software Engineering, ICSE '18. ACM, New York, pp 151–162, https://doi.org/10.1145/3180155.3180250

Tsantalis N, Mazinanian D, Rostami S (2017) Clone refactoring with lambda expressions. In: Proceedings of the 39th International Conference on Software Engineering, ICSE '17. IEEE Press, Piscataway, pp 60–70, https://doi.org/10.1109/ICSE.2017.14

Tufano M, Palomba F, Bavota G, Oliveto R, Di Penta M, De Lucia A, Poshyvanyk D (2015) When and why your code starts to smell bad. In: Proceedings of the 37th International Conference on Software Engineering - Volume 1, ICSE '15. IEEE Press, Piscataway, pp 403–414

Wang X, Sun J, Chen Z, Zhang P, Wang J, Lin Y (2018) Towards optimal concolic testing. In: Proceedings of the 40th International Conference on Software Engineering, ICSE '18. ACM, New York, pp 291–302, https://doi.org/10.1145/3180155.3180177

Waterman M, Noble J, Allan G (2015) How much up-front?: A grounded theory of agile architecture. In: Proceedings of the 37th International Conference on Software Engineering - Volume 1, ICSE '15. IEEE Press, Piscataway, pp 347-357

Yan H, Sui Y, Chen S, Xue J (2018) Spatio-temporal context reduction: a pointer-analysis-based static approach for detecting use-after-free vulnerabilities. In: Chaudron M, Crnkovic I, Chechik M, Harman M (eds) Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018. ACM, pp 327–337. https://doi.org/10.1145/3180155.3180178

Ye X, Shen H, Ma X, Bunescu R, Liu C (2016) From word embeddings to document similarities for improved information retrieval in software engineering. In: Proceedings of the 38th International Conference on Software Engineering, ICSE '16. ACM, New York, pp 404–415, https://doi.org/10.1145/2884781.2884862

Yu T, Qu X, Cohen MB (2016) Vdtest: an automated framework to support testing for virtual devices. In: Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016, pp 583–594. https://doi.org/10.1145/2884781.2884866

Emelie Engström is a Senior Lecturer at Lund University, Sweden, and a member of the Software Engineering Research Group. She conducts research in collaboration with industry and in national and international research collaborations. Her research interests include empirical software engineering, knowledge building and communication between industry and academia in software engineering, and decision support for software testing. She serves as a reviewer for several software engineering journals and conferences.

Margaret-Anne Storey is a Professor of Computer Science at the University of Victoria. She holds a Canada Research Chair in Human and Social Aspects of Software Engineering. She seeks to understand how software tools, communication media, data visualizations, and social theories can be leveraged to improve how software engineers and knowledge workers explore, understand, analyze and share complex information and knowledge. Over the past several years, she has collaborated with product teams and researchers at Microsoft to understand developer satisfaction and developer productivity, with the goal of improving their engineering systems and processes.

Dr. Per Runeson is a professor of software engineering at Lund University, Sweden, and the leader of its Software Engineering Research Group (SERG). His research interests include empirical research on software development and management methods, in particular for software testing and open innovation, and cross disciplinary topics on the digital society. He has published more than 150 papers and contributed significantly to software engineering research methodology by the books on case studies and experimentation in software engineering. He serves on the editorial boards of Empirical Software Engineering and Software Testing, Verification and Reliability, and is a member of several program committees.

Martin Höst is a Professor in Software Engineering at Lund University, Sweden. He received an M.Sc. degree from Lund University in 1992 and a Ph.D. degree in Software Engineering from the same university in 1999. His main research interests include open source, security, IT vulnerability, and software quality. The research is mainly conducted through empirical methods such as case studies, controlled experiments, and surveys. He has published more than 90 articles in international journals and proceedings from conferences and workshops.

Maria Teresa Baldassarre is associate professor at the University of Bari, Italy, and member of the Software Engineering Research Laboratory (SERLab). Her research interests are: empirical software engineering, human factors in software engineering, quality assessment, improvement and management in software processes, products and projects. She collaborates on several research projects and carries out controlled and in field experimentation within small and medium enterprises. She is a partner of the SER&Practices spin off company of the University of Bari. She is actively involved in research projects and collaborations with international partners. Currently she is the representative of the University of Bari in the International Software Engineering Research Network (ISERN), and is involved in various program committees related to relevant software engineering and international empirical software engineering venues.

Affiliations

Emelie Engström[1] · Margaret-Anne Storey[2] · Per Runeson[1] · Martin Höst[1] · Maria Teresa Baldassarre[3]

Margaret-Anne Storey
mstorey@uvic.ca

Per Runeson
per.runeson@cs.lth.se

Martin Höst
martin.host@cs.lth.se

Maria Teresa Baldassarre
mariateresa.baldassarre@uniba.it

[1] Lund University, Lund, Sweden

[2] University of Victoria, Victoria, BC Canada

[3] University of Bari, Bari, Italy