Noname manuscript No.
(will be inserted by the editor)

A review of software engineering research from a design science perspective

Emelie Engström · Margaret-Anne Storey · Per Runeson · Martin Höst · Maria Teresa Baldassarre

Received: date / Accepted: date

Abstract *Background:* Communicating software engineering research to industry practitioners and to other researchers can be challenging due to its context dependent nature. Design science is recognized as a pragmatic research paradigm, addressing this and other characteristics of applied and prescriptive research. Applying the design science lens to software engineering research may improve the communication of research contributions. *Aim:* The aim of this study is to 1) evaluate how well the design science lens helps frame software engineering research contributions, and 2) identify and 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 selected set of 38 top software engineering publications to extract and analyze their design science contributions. *Results:* We identified five clusters of papers, classified based on their alignment to the design science paradigm. *Conclusions:* The design science lens helps to pinpoint the theoretical contribution of a research output, which in turn is the core for assessing the practical relevance and novelty of the prescribed rule as well as the rigor of applied empirical methods in support of the rule.

Keywords Design science · Research review · Empirical software engineering

Emelie Engström, Per Runeson, Martin Höst
Lund University, Sweden
E-mail: [emelie.engstrom,per.runeson,martin.host]@cs.lth.se

Margaret-Anne Storey
University of Victoria, Canada
E-mail: mstorey@uvic.ca

Maria Teresa Baldassarre
University of Bari, Italy
E-mail: mariateresa.baldassarre@uniba.it

1 Introduction

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 at designing solutions to practical 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.

Design science is an accepted research paradigm in the fields of information systems and other engineering disciplines. 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 [40]. Although Wierenga *et al.* have promoted design science for capturing design knowledge in software engineering [52], we seldom see it being referred to in our field (although there are some exceptions such as [53]). 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 [41].

The goal of our research is to investigate if and how the design science paradigm may be a viable way to present 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 [48].

We inspected 38 ACM distinguished papers published at 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 the research contributions communicated in these papers. Also, most of the papers described research contributions that are congruent with the design science paradigm, even though none of them explicitly used the term. Applying this lens furthermore 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, but also how the communication of their research contributions could be further improved for practitioners and researchers to utilize and build on these contributions respectively.

We advocate that the design science lens not only helps relate and synthesize technical and theoretical knowledge about similar problems and solutions, but that doing so may help us present research in a way that more easily communicates our research findings to industry and that may lead to faster adoption and validation of the technical solutions our community proposes.

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. We then describe our methodology for generating visual abstracts for the cohort of ACM distinguished papers we studied and use the information highlighted by the abstracts to extract the design knowledge in each paper. Finally we cluster the papers by the design knowledge produced. We interpret and discuss the implications of our findings, outline the limitations of our study and discuss related work before concluding the paper.

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 workshops on the topic. This work helped us develop a visual abstract template to use as a lens for communicating and assessing research contributions [48]. In this section, we summarize the findings by giving 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 in software engineering

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 Figure 1 we illustrate the relationship between *problem formulation* and *solution proposal*, as well as between *theory* and *practice*. The arrows in the figure represent different types of contributions of design science research, i.e., problem characterization, solution design, instantiation, abstraction, and validation.

Design knowledge is holistic and heuristic by its nature, and must be justified by in-context validations [52, 1]. The term holistic is used by van Aken [1] 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 [16]. As a consequence, we can never prove the effect of a solution conclusively, but need to 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* [1], which are rules that capture general knowledge about the *mappings* between *problems* and proposed *solutions*.

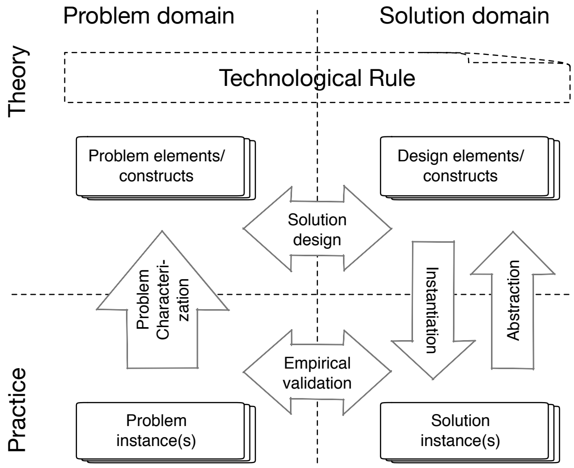

Fig. 1 An illustration of the interplay between problem and solution as well as between theory and practice in design science research. The arrows illustrate the knowledge-creating activities and the boxes represent the levels and types of knowledge that is created.

Van Aken describes the typical design science strategy to be the multiple case study [1], 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 work or not work. Each new case adds insights that can refine the technological rule until saturation is achieved [1]. Gregor and Hevner present a similar view of knowledge growth through multiple design cycles [21]. Wieringa [51] and Johannesson [26] discuss action research as one of several empirical methodologies that can be used to produce design knowledge. 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 The design science visual abstract template

The visual abstract template, as presented in Figure 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* and *assess* design knowledge (i.e., 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

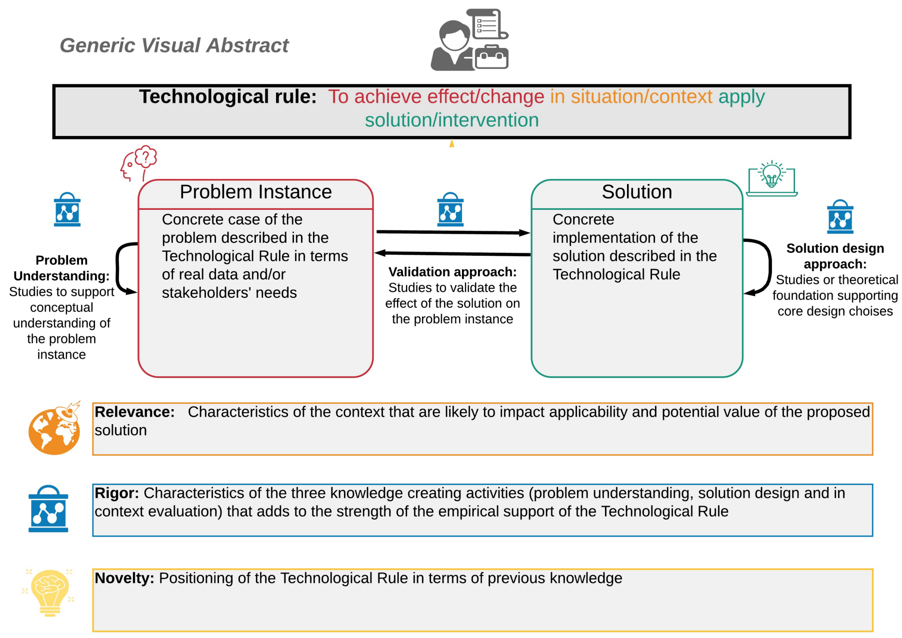

Fig. 2 The visual abstract template [48] used for paper analysis. It captures three main aspects of design science contributions: 1) the theory proposed or refined in terms of a technological rule; 2) the empirical contribution of the study in terms of one or more instances of a problem-solution pair and corresponding design and validation cycles; and 3) the assessment of the value of the produced knowledge in terms of relevance, rigor, and novelty.

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), articulating both.

2.2.1 The technological rule

In line with van Aken [1], our visual abstract template emphasizes technological rules 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 understanding*, *solution design*, or *solution validation*, applied in each instantiation.

Technological rules can be expressed at any convenient abstraction level and are hierarchically related to each other. However, technological rules expressed at a very 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), while rules at very low abstraction levels have a narrow scope and thus lack relevance for most software engineers. Thus, 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 empirical contribution in support of the rule

The main body of the visual abstract template 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 understanding, solution design and validation.

2.2.3 Assessing the contribution

The ultimate goal of design science research is to produce general design knowledge rather than to solve the problems of the unique instances. Thus the value of the research should be assessed with respect to the technological rule (i.e. design knowledge) produced. The information in the three assessment boxes aims at aiding the reader in making an assessment that is relevant for their context.

The relevance box aims to support answering the question *To whom is this technological rule relevant?* 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 is 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 the studied problem is. To enable both types of assessment, relevant context factors need to be reported as discussed for example. A taxonomy of context factors in software engineering was proposed by Petersen and Wohlin [34]. However, as discussed by Dybå et al. [16], not all context factors are helpful in making this assessment. Only those that are critical for either the applicability of the solution or for the potential effect of applying a solution should be reported.

The rigor box aims to support answering the question *How mature is the technological rule?* Rigor may be assessed with respect to all of the three knowledge-creating activities: problem understanding, solution design and solution validation. However, solution design is a creative process by nature and 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. On the other hand, the other two activities, problem understanding and solution validation, are based on common empirical methods on which relevant validity criteria (e.g., construct validity) can be applied.

The novelty box aims at capturing the positioning of the technological rule in terms of previous knowledge and 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 general relevance.

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. Replication of experiments has been discussed [13, 27, 42] and is encouraged[1] by the software engineering community. While experimental replications may be *exact* replications in which the procedures of an experiment are followed as closely as possible, the incremental adding of empirical support for a technological 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.* [42].

3 Methodology

The main goal of this paper is to investigate how well software engineering (SE) research contributions are aligned with the design science paradigm.

As part of this work, we aim 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 [48]: the left side of Figure 3 shows the steps we followed to arrive at the initial version of the visual abstract template, while the right side shows the steps we followed in the research reported in this paper.

[1] https://2018.fseconference.org/track/rosefest-2018

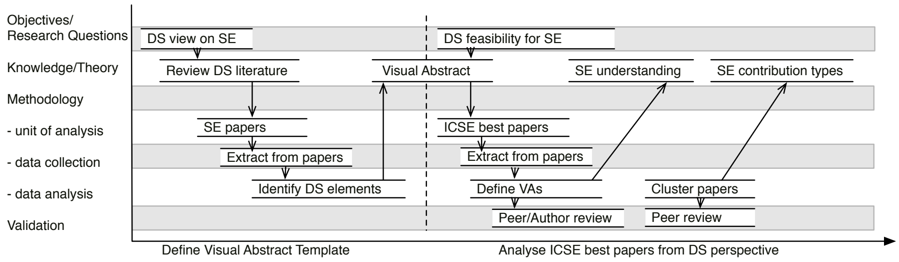

Fig. 3 The left side of this figure shows the approach we followed to develop the initial version of the visual abstract, while the right side shows the main steps of the research presented in this paper which also helped us refine the visual abstract template and instructions for filling it out.

We used this visual abstract template to describe the research contributions in a particular set of papers from the ICSE conference: those that were selected as the top 10% of papers 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 contributions across 38 papers.

The process for defining the visual abstracts was as follows. Each paper, in the cohort of ICSE distinguished papers from 2014–2018, was randomly assigned to two reviewers among the authors of this paper. The two reviewers independently answered the set of *design science questions* listed in Table 1. This set of questions map to the different components in the visual abstract template. For each paper we iterated until we arrived at an agreement for a shared response to these questions, seeking additional input from the rest of our research team and seeking expert opinions for papers on topics unfamiliar to us.

The answers to the questions were captured in a spreadsheet to facilitate future analysis and ongoing review and internal auditing of our process. Our combined responses were then used to populate the visual abstract template for each paper. The collection of visual abstracts for all of the papers is available online at `dsse.org`.

As part of our analysis, we confirmed our interpretations of the 2014 ICSE best papers with the original authors. We heard back from half of the authors, who confirmed the accuracy of our responses (mentioning minor improvements only). However, this was a time-consuming task that placed a burden on the original paper authors, so we did not repeat this step for the other papers—although the abstracts for all papers we studied are available online and those authors may comment publicly on our interpretations if they choose.

When we had confirmed the visual abstracts for all of the papers, we began the clustering of papers. We reexamined the papers (working in the same pairs

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?

as the first phase) and answered a second set of questions (see Table 2). Note as we answered questions in Tables 1 and 2, we presented our answers to other members in our research group for feedback which in many cases helped us refine those 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 to identify *clusters* representing different types of design science contributions. The answers to the meta-analysis questions in Table 2 helped us cluster the papers according to problem understanding, solution and validation contributions, as well as consider the novelty of the theoretical contributions, the relevance of the work, and aspects of the solution design process.

Following our face-to-face visual abstract sorting activity, we worked again in pairs to inspect each of the papers in the clusters in order to confirm whether we had categorized them correctly. Again, 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

Table 2 Meta-analysis: The answers to the following questions were used in the paper clustering step of our research process to group papers by design knowledge contribution

1.	*Type of paper*
1.1	Which type of study does the paper best fit into: Design Science, Explanatory (Descriptive) Science?
1.2.	What is the main focus of the paper? Problem understanding, Solution design, validation of Solution to Problem?
2.	*Theory beyond the instance level*
2.1	Is the technological rule clearly defined in the paper as a generalization of the instance-level problem-solution pair?
2.2	Is the technological rule a new proposal/a validated existing rule/a refinement of an existing rule/an existing rule debunked by the study?
2.3	Is the technological rule aimed at researchers or practitioners?
3.	*Relevance*
3.1	Is relevance explained: is it easy for a practitioner to assess relevance of the rule for their case?
3.2	Are stakeholders involved in the study? Are we aware of any practitioners asking for the solution?
4.	*Design*
4.1	Are alternative solutions considered?

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 were better framed through an explanatory lens as the investigated problems were not linked to a specific 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 addressed the questions in Table 2.

1. *Problem solution pair*: this cluster represents papers that equally balance their focus on 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 understanding.
3. *Solution design*: papers in this cluster focus on the design of the solution rather than on problem understanding 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 research insights for researchers rather than for practitioners.

Figure 4 illustrates how the first four clusters (1-4) map to the design science view, including both the problem-solution dimension and the general-specific one. Clusters 1-3 all represent different types of design science research

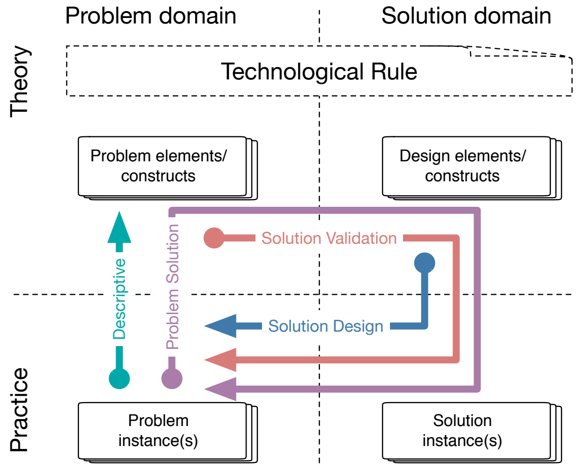

Fig. 4 An illustration of how the identified clusters map to the problem/solution and the general/specific axes respectively. The arrows show how typical studies in each cluster traverse the four quadrants (1. practical problem, 2. conceptual problem description, 3. general solution design, and 4. instantiated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

since the papers in these clusters consider explicit problem solution pairs. Papers in the fourth cluster provide 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 on software engineering research rather than on software engineering practice.

Figure 5 shows 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.

A more detailed and nuanced description for each cluster is provided below. For each cluster we refer to examples of papers and include one visual abstract 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

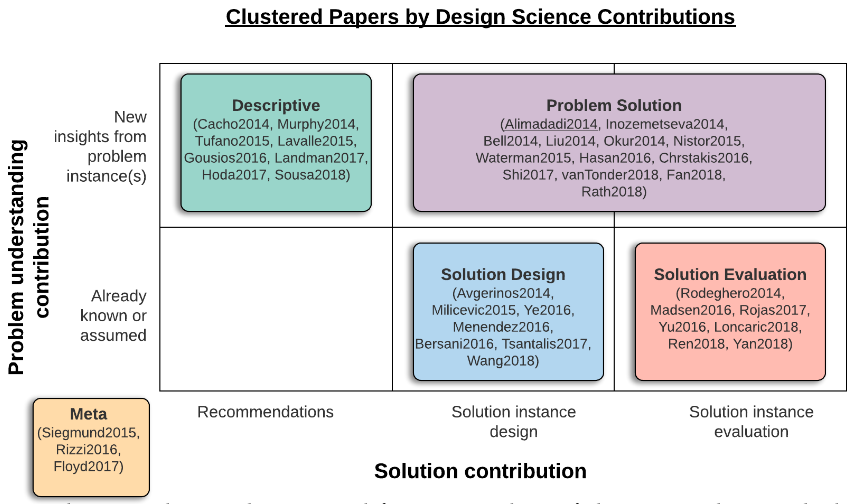

Fig. 5 The main clusters that emerged from our analysis of the papers, showing the key design science contributions in terms of problem understanding insights and solution recommendations, design and/or validation.

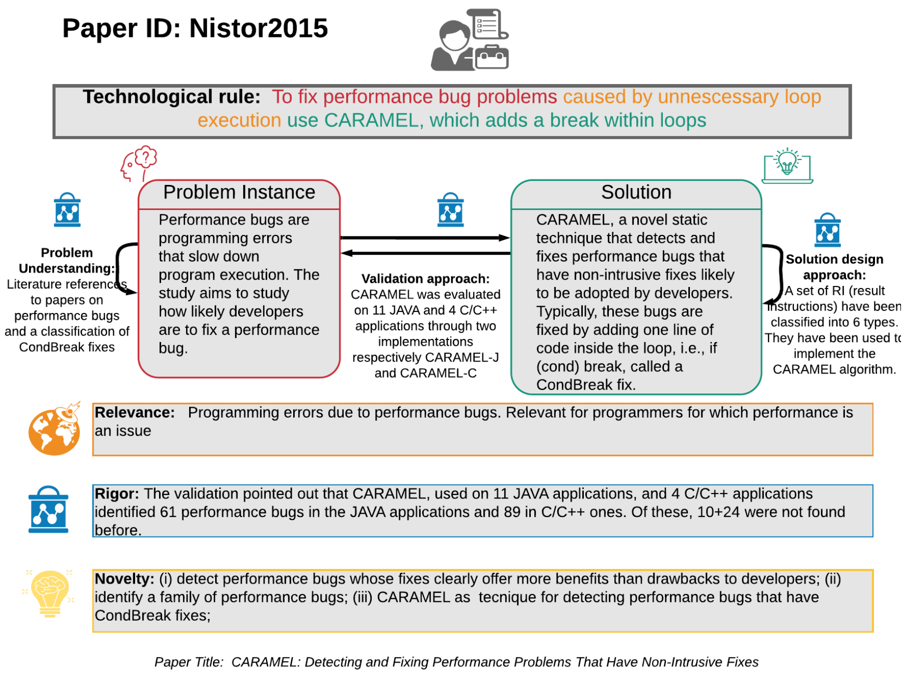

Fig. 6 Visual abstract of a typical paper in the problem solution cluster, Nistor et al. [39]

that many software engineering papers can be framed in accordance with the 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 to adopt to solve the problem instance (see Figure 6). Consequently, the relation among problem (e.g., performance bug problems) and solution (e.g., novel static analysis technique CARAMEL that detects and fixes performance bugs) 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 [31] or controlled experiments [4, 7].

In some cases alternative solutions are compared to the proposals made. For example, Rath *et al.* [36] considered alternative information retrieval techniques and classifiers during the design of their solution, and used precision/recall values collected from all the compared solutions to develop their classifier.

A representative example for this cluster is the paper by Nistor *et al.* [31]. Given the problem instance, where analysis of the related literature points out that performance bugs are programming errors that slow down program execution, the authors investigate how likely developers are to fix a performance bug. The solution proposed is a novel static analysis technique, CARAMEL, which is able to detect and fix performance bugs. Nistor *et al.* designed a set of case studies to validate the tool on a set of Java and C++ applications. The visual abstract is shown in Figure 6. Other visual abstracts in this cluster (and other clusters) are available on our online website.

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 (in many cases 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”.

As a typical example, Loncaric *et al.* [29], 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

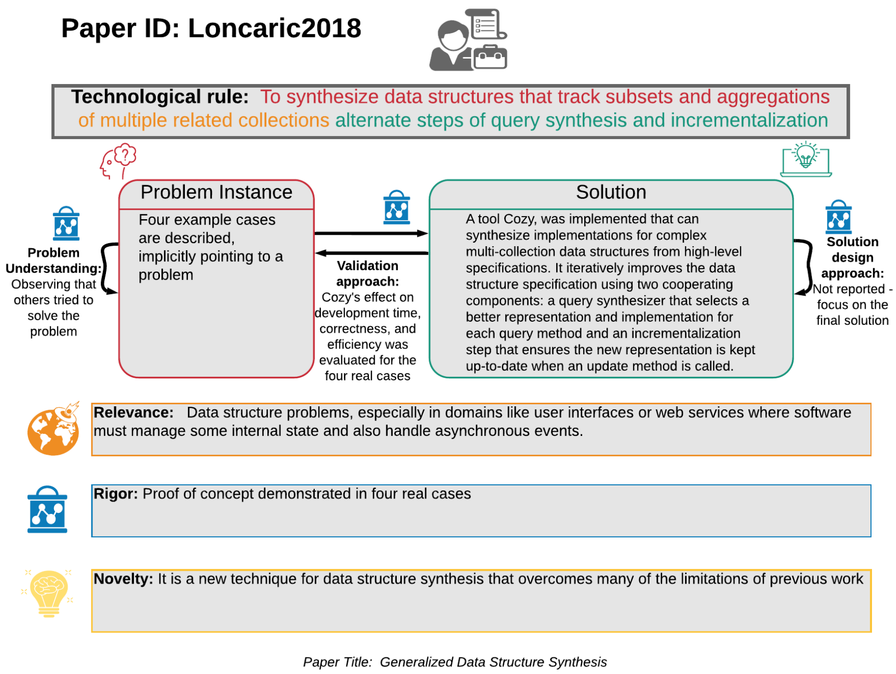

Fig. 7 Visual abstract of a typical paper in the cluster of solution validation studies, Loncaric *et al.* [29]

concept is demonstrated in four real cases. The corresponding visual abstract is presented in Figure 7.

Note that some 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 one. For example, Rodeghero *et al.* [38] 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 and 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.* [39] 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,

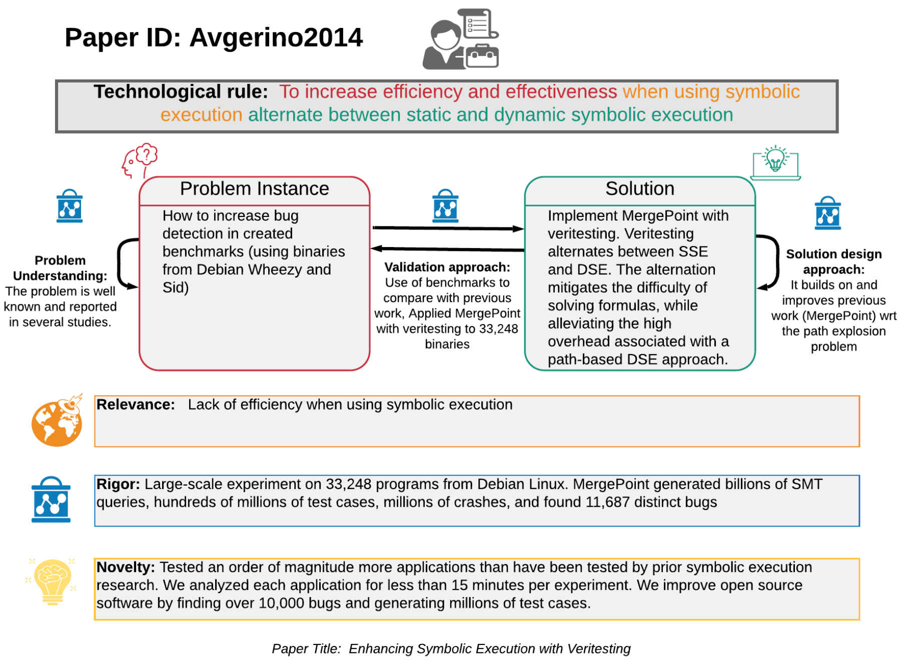

Fig. 8 Visual abstract of a typical paper in the solution design cluster, Avgerinos et al. [5]

used by Yan *et al.* [54] to motivate addressing the vulnerability problem of Use-After-Free pointers.

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.* [5] present a new way of testing with symbolic execution, see Figure 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, in Bersani *et al.* [8], these authors 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, i.e., a new trace checking algorithm. A similar observation can be made for analysis and validation, i.e., the analysis in Avgerinos *et al.* [5] 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.* [8], the validation is carried out comparing the designed solution with other, point-based semantics.

For papers in this clusters, the technological rule 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 typically conducted by applying the proposed solution on a code base and analyzing metrics of, e.g., the number of faults found in testing. Typically, 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, papers in the solution design category focus mainly on the presented solution as the main contribution. However, the solution is often validated on a data set in order to demonstrate its validity with respect to a problem that is understood by the reader.

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 *et al.* [30] 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 *et al.* [24] 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.*[20] surveyed 4000 open source contributors to understand the pull-based code contribution process, Tufano *et al.* [49] analyzed git commits from 200 open source repositories to investigate more about code smells, Cacho *et al.* [11] studied changes to 119 versions of code extracted from 16 different projects to understand trade-offs between robustness and maintenance and Lavallee *et al.* [28] 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 explanatory problem characterizations. In four papers out of eight, a list of recommendations is provided

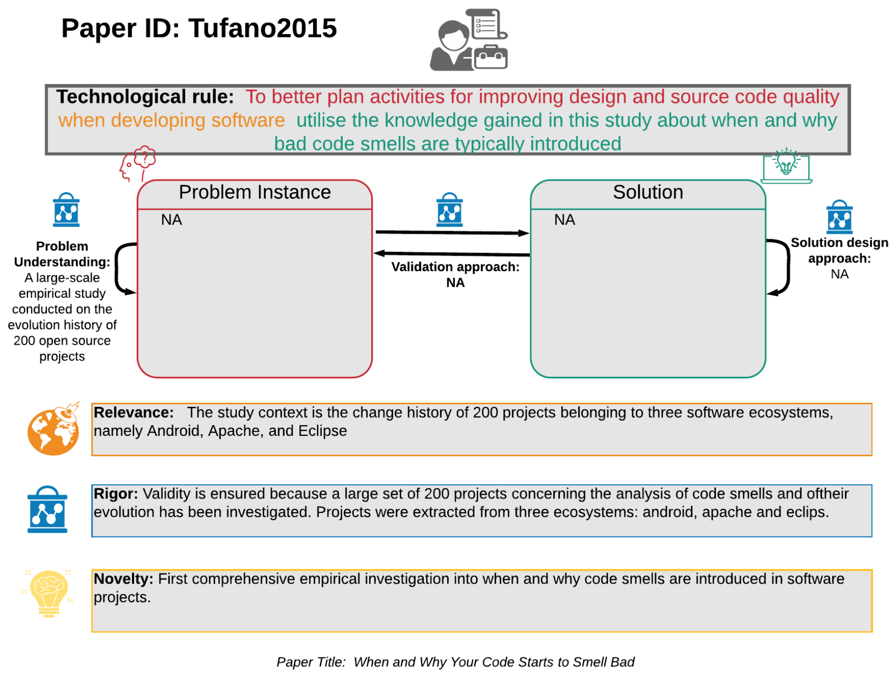

Fig. 9 Visual abstract of a typical paper in the cluster of descriptive studies, Tufano et al. [49]

as well. Thus, it is in most cases possible to derive several technological rules from one 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.* [20] 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.* [28] pinpoint the lack of knowledge about the impact of organizational factors on software quality). Novelty or positioning is, on the other hand, not described in terms of the problem or the solution but about aspects of the study as a whole. Gousios *et al.* [20] add a *novel perspective*, the contributors' in code review, Lavalle *et al.* [28] add *more empirical data* about organizational factors and software quality, and Tufano *et al.* [49] claim to report the *first empirical investigation* of how code smells evolve over time.

In summary, although the descriptive papers may contribute to design knowledge, i.e., problem understanding 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

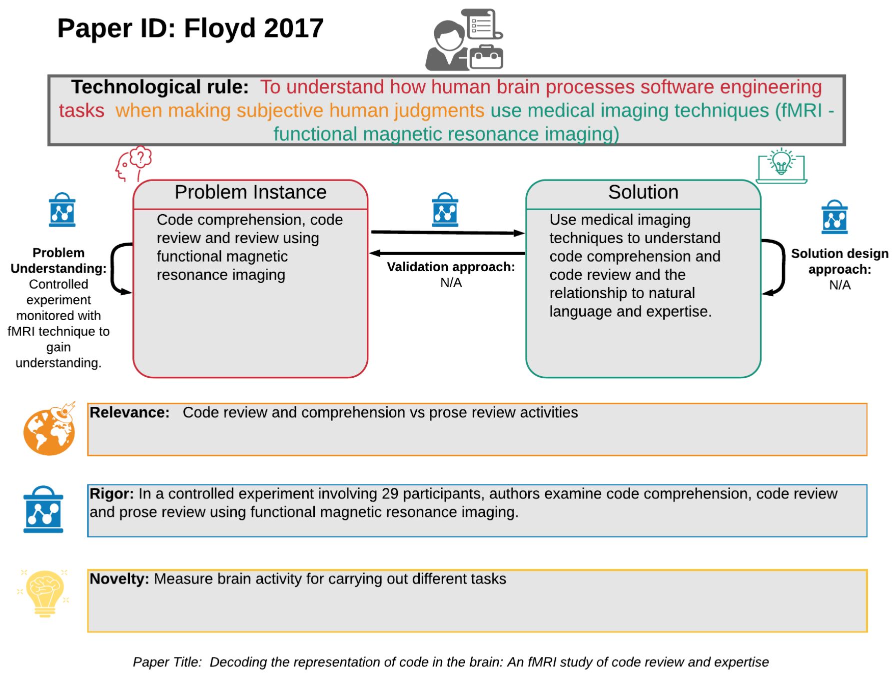

Fig. 10 A typical example of a visual abstract in the Meta cluster, Floyd et al. [18]

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 have other applications 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 papers in the population we considered do not aim to identify or solve real world software engineering problems, but rather they are aimed at identifying or solving problems software engineering *researchers* may experience. Siegmund *et al.* [43] conduct a study that reveals how the software engineering research community lacks a consensus on internal and external validity. Rizzi *et al.* [37] advise researchers how to improve the efficiency of tools that support large-scale trace checking. Finally, Floyd *et al.* [18] propose how fMRI methods can help software engineering gain more insights on how developers comprehend code, and in turn may improve comprehension activities.

In summary, some 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. We show the visual abstract for the Floyd *et al.* [18] paper in Figure 10. The Meta category of papers may show relevance to industry but in an indirect manner.

5 Interpretation of results

The long term goal of software engineering research, in our view, is to address real world, large scale problems, and provide useful recommendations with evidence for their benefits, as well as acknowledging their weaknesses. The implications for research of this view are (at least) twofold: (i) the research must be conducted with real problems in mind, and (ii) the communication between industry and academia must be adapted from the style of communication in purely academic fields. These issues are discussed in several papers, referring to the 'gap' between research and practice, e.g., recently by Gouses et al. in IEEE Software's special edition on Software Engineering's 50th anniversary [19]. This study indicates that the design science lens may contribute to addressing both these aspects.

5.1 Design science in software engineering

We have, as discussed in Section 2.1, found the design science paradigm to be well aligned with the needs in software engineering research. *Problem understanding* activities may help conceptualize instances of complex, socio-technical system of systems. The *design knowledge* generated in the research is not limited to the concrete solution instance of the research endeavor, but may be generalized to other contexts as well, thereby providing theoretical contributions to the field of software engineering. The *validation* of the problem–solution pair brings transparency and objectivity to the research, resonating with our aim to move away from advocacy research. Empirical research methods – quantitative as well as qualitative – fit well into both problem understanding and validation.

Although we consider much of the software engineering research to be design science research, in that its long term goal is to provide well-informed recommendations to practitioners, our analysis demonstrates that not all exploratory and descriptive research provides relevant design knowledge (see the Descriptive cluster in Section 4.4). If the descriptive knowledge is to be considered design knowledge, it must be described in terms of the envisioned solution.

Communication is a key challenge in industry-academia collaboration. The means of communication on academia – 10 page small font papers in academic style – is very different from industry's slide deck communication style. Our proposed Visual Abstracts template [48] (see Figure 2) addresses this challenge, based on the design science paradigm. Particularly, the *technological rule* is a condensed expression, linking the instance of a problem-solution pair with a generalized recommendation.

Our analysis of ICSE best papers reveals design science contributions in terms of *problem understanding*, *solution design and implementation*, and *solution evaluation*, as depicted in Figure 4 (RQ1). In some papers, all four elements are explored in equal depth, while other papers focus on two or three elements, as presented in the clustering above in Section 4. However, none of the papers is *presented* in terms of these elements and we had to spend significant amount of effort, using the questions in Table 1 to extract the knowledge in a systematic way.

We also extracted *technological rules* from the papers. None of the papers had any conclusion or recommendation in such a condensed form (RQ2). In some cases, the abstracts and introduction sections were written clear enough to 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 [10], but not in ICSE. Introducing technological rules would, we believe, help in communicating the core of the contribution, both to peer academics and to industry. Development towards more explicit theory building in software engineering [44, 46] may also pave the way for technological rules as a means to express theoretical contributions.

5.2 Assessing design knowledge contributions

The rigor in the research steps of the studied papers is generally high, as they are awarded best papers in the competitive context of ICSE (RQ3). However, depending on the cluster, different aspects of the design knowledge are addressed with the most rigor: some report a systematic approach to *problem understanding*, while others stress the rigorous evaluation of the design.

Solution validation and *solution design* papers tend to rigorously benchmark their solution against a code base or other artifacts to demonstrate the merits of the proposed approach. Validating the solutions in industrial contexts are less prevalent in these sets of papers. This may partially explain that we observed some lack of explicit relevance analysis from stakeholder perspectives in the solution design and solution validation clusters.

Research is of risk of sub-optimizing solutions, if built on top of previous research, making improvements in relation to earlier solutions, and positioning the contribution as novel or rigorous from that perspective (RQ3). Research benchmarks, e.g., code bases or data sets, help improve the rigor of the validation of detailed technological rules, but are at the same at risk of making the outcomes less relevant for practitioner stakeholders as the validation of the general technological rule is disregarded. An example of such risks was uncovered in the field of improving information retrieval (IR) algorithms for trace recovery. In their analysis of 25 papers, optimizing the precision and recall of trace recoveries, Borg and Runeson conclude “Based on the primary studies

... there is no evidence of any IR model consistently outperforming another. On the contrary, we see a clear interaction between IR model and the data used for the empirical validation.” [9].

Descriptive studies are not considered design science unless they are conducted with an envisioned solution in mind. However, the methodological rigor is quite straightforward to assess for these studies, as there are standards of rigor, which have been discussed in software engineering communities. However, when it comes to balancing the need for rigor between the design science elements of problem understanding, solution design and validation, research communities have to make this discussion explicit. Interestingly enough, one of the papers in the *meta* cluster discusses balancing different aspects of validity [43], and rigorousness in reporting experimental setups is reported in another meta paper [37].

Many researchers and practitioners that engage in our research community feel that the impact of much of our software engineering research on industry is not as high as we may expect. Presenting our research through a design science lens can perhaps better highlight problem insights and design knowledge and could perhaps help in reaching the ears and eyes of practitioners with relevant contributions.

5.3 Recommendations for software engineering research

As researchers (and reviewers) ourselves we find that contributions from research papers are often not evident, and thus reviewers (and practitioners) may miss the value in the papers. Currently, templates for reviewing papers can even be biased towards just a few aspects, e.g., solution design or rigor (at the expense of relevance, for example). We therefore propose using the design science lens as one additional way to present and assess software engineering research. To further advance the practice in our community, we recommend:

– *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 [48], could help in communicating the research contributions to practitioners and peer researchers. Expressing the technological rules clearly and at a carefully selected level of abstraction, 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 the solution. Without an explicit problem instance, the research is at risk of losing connection with the original question, as researchers dig deeper and deeper in to details of a particular intervention.
– *Choose validation methods and context consciously*: The choice of methods and context for the validation may be different, depending on the intended scope of the theoretical contribution. Therefore, validation methods and contexts should be consciously chosen to represent the perspective of relevant stakeholders.

– *Use the design science lens as a 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 context. Similarly, the design science perspective can be used as an analysis tool in mapping studies [33], to assess existing research and identify research gaps.
– *Consider research design as a design science*: The cluster of *meta* studies, which are primarily aimed for researchers, indicate that the design science lens also fits for the design and conduct of research studies. The papers address problems in conducting research and propose solutions to help achieve higher quality research contributions. Conducting and presenting these in the same way as studies in the software engineering domain adds to their credibility.

We hypothesize that following these recommendations, based on our in depth analysis of ICSE best 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 conferences and journals.

6 Limitations

In order to understand how design science can be used as 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 fine exemplars of SE research. We acknowledge that we would likely see a different result in terms of the numbers of papers for a different population of papers (e.g., all papers presented at ICSE or in other venues or journals). That said, we purposefully selected this sample of papers as a exploratory step in our research and don't claim our result would generalize.

Our view of design science may not match other views that are reported in the literature. We developed our view from examining several interpretations of design science as discussed in [48] and in Section 2. Our view was developed over the course of two years spent reading and discussing many design science papers; our interpretation was developed in an iterative manner. We have used our visual abstract template in several workshops and received favourable feedback about the viable application of the template to software engineering papers that contain design knowledge.

We recognize that our interpretations of the research contributions from the papers we examined may not be entirely accurate or complete. As mentioned earlier in the paper, initially we intended to seek feedback from the authors of all the papers to check that our view of the design knowledge in their papers was accurate, and indeed we sent a batch of papers (from 2014) for review by the authors. However, we found the response rate was rather low (only 7 of 14 of the selected set of paper authors responded). For those that did respond, all but one agreed with our summaries presented through the visual abstracts, while this sole initial disagreement was due to misinterpretation of the visual abstract template. This feedback served as some validation that we were proceeding in the right direction. Consequently, we decided to rely on our judgment. To do so, we divided papers equally among all the authors assigning two to each paper. They would independently answer the design science questions (as mentioned in Section 7), then refer back to the paper in cases of disagreement, and merge our responses until we reached full agreement. Following cases of existing disagreement, we sought additional expert opinions. Finally, we reviewed all of the abstracts as a group to reconfirm our interpretation. These abstracts are available online and open for external audit by the authors or others in the community.

To derive clusters of the papers, we followed quite a rigorous process. We met face to face in a several hour workshop and followed up in several sessions over several months 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 `dsse.org`. We welcome comments on our clusters and the categorization of individual papers.

7 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 investigate its usefulness in the software engineering context. 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. Furthermore, we present other initiatives that aimed to improve communication and dissemination of software engineering research contributions.

7.1 Design science conceptualizations

Design science has been conceptualized by several researchers in software engineering [50] and other disciplines, such as information systems [23], [21], [26] and organization and management [2]. Wieringa describes design science as an act of producing knowledge by designing useful things [50] and makes a distinction between knowledge problems and practical problems. Similarly, Gregor and Hevner emphasize the dual focus on the artifact and its design [21] 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 this 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 artefacts such as tools, models and techniques or distilled to simple technological rules. In line with van Aken [2], 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 Stol and Fizgerald's distinction between knowledge seeking research and solution seeking research [47]. 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 [2]. A corresponding division of software engineering theory has not been proposed yet, although theory types are discussed by software engineering researchers [44], [45].

In the area of Information Systems, several literature reviews were conducted of design science research. Indulska and Recker [25] 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. [15,14] have 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 have 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.

7.2 Communication of research results

Goues et al. discuss how research results can be presented in a more useful way to practitioners [19]. They propose, for example, that systematic literature reviews should make recommendations to practice and that the concept of evidence should be understood in the same way by practitioners and research through a framework. That is, they propose work towards consensus on a framework for evidence and mapping between evidence and recommendation. They also propose explicit identification of results to practitioners, and they emphasize the importance of recognizing and having incentives for reviews and meta analysis in order to synthesize actionable guidance. Petersen and Engström developed a framework, SERP taxonomy architecture [32] including the constructs of a technological rule (i.e. desired effect, context and intervention), to support the mapping of practical problems with research. This framework has then been used to develop a taxonomy to establish a common understanding between practitioners and researchers in software testing [17] and to review regression testing literature from a relevance point of view [3] leading to generalized recommendations in terms of technological rules.

Concerning relevance, Beecham et al. communicated with a test group of practitioners [6] 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. [35] in a study based on focus groups with practitioners as well as publications. 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. [22], who conclude that practitioners assess the usefulness of many artifacts as being too low. This is inline 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 thus far.

Another attempt to make evidence available to practitioners is presented by Cartaxo et al. [12]. 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 the by both practitioners and researchers. While evidence briefings may provide an effective way to synthesize evidence from several studies our visual abstract template provides a means to effectively summarize the contribution of one study or research program from a design science perspective.

8 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 many of these papers can be expressed in terms of the 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 the 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 — 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 (this was not a goal of this current work). In the meantime, we anticipate that our analysis of ICSE best 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.

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 for trying out the visual abstract on some of the ICSE best papers. Their enthusiasm encouraged us to continue with this research. The research was partially funded by Lund University, faculty of engineering for the Lise Meitner guest professorship (Storey), the ELLIIT strategic research area (Engström), and the EASE industrial excellence center (Runeson).

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: Paradigm of the Design Sciences. Journal of Management Studies **41**(2), 219–246 (2004). DOI 10.1111/j.1467-6486.2004.00430.x
2. van Aken, J.E.: Management Research as a Design Science: Articulating the Research Products of Mode 2 Knowledge Production in Management. British Journal of Management **16**(1), 19–36 (2005). DOI 10.1111/j.1467-8551.2005.00437.x
3. Ali, N.B., Engström, E., Taromirad, M., Mousavi, M.R., Minhas, N.M., Helgesson, D., Kunze, S., Varshosaz, M.: On the search for industry-relevant regression testing research. Empirical Software Engineering (2019). DOI 10.1007/s10664-018-9670-1

4. Alimadadi, S., Sequeira, S., Mesbah, A., Pattabiraman, K.: Understanding javascript event-based interactions. In: Proceedings of the 36th International Conference on Software Engineering, ICSE 2014, pp. 367–377. ACM, New York, NY, USA (2014). DOI 10.1145/2568225.2568268
5. Avgerinos, T., Rebert, A., Cha, S.K., Brumley, D.: Enhancing symbolic execution with veritesting. In: Proceedings of the 36th International Conference on Software Engineering, ICSE 2014, pp. 1083–1094. ACM, New York, NY, USA (2014). DOI 10.1145/2568225.2568293
6. Beecham, S., O'Leary, P., Baker, S., Richardson, I., Noll, J.: Making Software Engineering Research Relevant. Computer **47**(4), 80–83 (2014). DOI 10.1109/MC.2014.92
7. Bell, J., Kaiser, G.: Unit test virtualization with vmvm. In: Proceedings of the 36th International Conference on Software Engineering, ICSE 2014, pp. 550–561. ACM, New York, NY, USA (2014). DOI 10.1145/2568225.2568248
8. Bersani, M.M., Bianculli, D., Ghezzi, C., Krstić, S., Pietro, P.S.: Efficient large-scale trace checking using mapreduce. In: Proceedings of the 38th International Conference on Software Engineering, ICSE '16, pp. 888–898. ACM, New York, NY, USA (2016). DOI 10.1145/2884781.2884832
9. Borg, M., Runeson, P.: IR in software traceability: From a bird's eye view. In: Proceedings Empirical Software Engineering and Measurements (ESEM), pp. 243–246 (2013)
10. Budgen, D., Kitchenham, B.A., Charters, S.M., Turner, M., Brereton, P., Linkman, S.G.: Presenting software engineering results using structured abstracts: a randomised experiment. Empirical Software Engineering **13**(4), 435–468 (2008). DOI 10.1007/s10664-008-9075-7
11. Cacho, N., César, T., Filipe, T., Soares, E., Cassio, A., Souza, R., Garcia, I., Barbosa, E.A., Garcia, A.: Trading robustness for maintainability: An empirical study of evolving c# programs. In: Proceedings of the 36th International Conference on Software Engineering, ICSE 2014, pp. 584–595. ACM, New York, NY, USA (2014). DOI 10.1145/2568225.2568308
12. Cartaxo, B., Pinto, G., Vieira, E., Soares, S.: 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 (2016)
13. Carver, J.C., Juristo, N., Baldassarre, M.T., Vegas, S.: Replications of software engineering experiments. Empirical Software Engineering **19**(2), 267–276 (2014). DOI 10.1007/s10664-013-9290-8
14. Deng, Q., Ji, S.: A review of design science research in information systems: Concept, process, outcome, and evaluation. Pacific Asia Journal of the Association for Information Systems **10** (2018)
15. Deng, Q., Wang, Y., , Ji, S.: Design science research in information systems: A systematic literature review 2001-2015. In: CONF-IRM 2017 Proceedings (2017)
16. Dybå, T., Sjøberg, D., Cruzes, D.S.: 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 (2012). DOI 10.1145/2372251.2372256
17. Engström, E., Petersen, K., Ali, N.B., Bjarnason, E.: Serp-test: a taxonomy for supporting industry–academia communication. Software Quality Journal **25**(4), 1269–1305 (2017). DOI 10.1007/s11219-016-9322-x
18. Floyd, B., Santander, T., Weimer, W.: 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, pp. 175–186. IEEE Press (2017)
19. Goues, C.L., Jaspan, C., Ozkaya, I., Shaw, M., Stolee, K.T.: Bridging the gap: From research to practical advice. IEEE Software **35**(5), 50–57 (2018). DOI 10.1109/MS.2018.3571235
20. Gousios, G., Storey, M.A., Bacchelli, A.: Work practices and challenges in pull-based development: The contributor's perspective. In: Proceedings of the 38th International Conference on Software Engineering, ICSE '16, pp. 285–296. ACM, New York, NY, USA (2016). DOI 10.1145/2884781.2884826
21. Gregor, S., Hevner, A.R.: Positioning and Presenting Design Science Research for Maximum Impact. MIS Q. **37**(2), 337–356 (2013)

22. Grigoleit, F., Vetro, A., Diebold, P., Fernandez, D.M., Bohm, W.: 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 (2015). DOI 10.1109/ESEM.2015.7321203
23. Hevner, A.R., March, S.T., Park, J., Ram, S.: Design science in information systems research. MIS Q. **28**(1), 75–105 (2004)
24. Hoda, R., Noble, J.: Becoming agile: a grounded theory of agile transitions in practice. In: Proceedings of the 39th International Conference on Software Engineering, pp. 141–151. IEEE Press (2017)
25. Indulska, M., Recker, J.C.: Design science in IS research : a literature analysis. In: S. Gregor, S. Ho (eds.) 4th Biennial ANU Workshop on Information Systems Foundations. ANU E Press, Canberra, Australia (2008)
26. Johannesson, P., Perjons, E.: An Introduction to Design Science. Springer Publishing Company, Incorporated (2014)
27. Juristo, N., Gómez, O.S.: Replication of software engineering experiments. In: Empirical software engineering and verification, pp. 60–88. Springer (2010)
28. Lavallée, M., Robillard, P.N.: 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, pp. 677–687. IEEE Press, Piscataway, NJ, USA (2015)
29. Loncaric, C., Ernst, M.D., Torlak, E.: Generalized data structure synthesis. In: M. Chaudron, I. Crnkovic, M. Chechik, M. Harman (eds.) Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018, pp. 958–968. ACM (2018). DOI 10.1145/3180155.3180211
30. Murphy-Hill, E., Zimmermann, T., Nagappan, N.: 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, pp. 1–11. ACM, New York, NY, USA (2014). DOI 10.1145/2568225.2568226
31. Nistor, A., Chang, P.C., Radoi, C., Lu, S.: 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, pp. 902–912. IEEE Press, Piscataway, NJ, USA (2015)
32. Petersen, K., Engström, E.: Finding relevant research solutions for practical problems: the serp taxonomy architecture. In: WISE@ASE (2014)
33. Petersen, K., Feldt, R., Mujtaba, S., Mattsson, M.: Systematic mapping studies in software engineering. In: Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering, EASE'08, pp. 68–77. BCS Learning & Development Ltd., Swindon, UK (2008)
34. Petersen, K., Wohlin, C.: Context in industrial software engineering research. In: Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM '09), pp. 401–404 (2009). DOI 10.1109/ESEM.2009.5316010
35. Rainer, A., Hall, T., Baddoo, N.: 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 (2003)
36. Rath, M., Rendall, J., Guo, J.L.C., Cleland-Huang, J., Mäder, P.: Traceability in the wild: Automatically augmenting incomplete trace links. In: Proceedings of the 40th International Conference on Software Engineering, ICSE '18, pp. 834–845. ACM, New York, NY, USA (2018). DOI 10.1145/3180155.3180207
37. Rizzi, E.F., Elbaum, S., Dwyer, M.B.: 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, pp. 132–143. ACM (2016)
38. Rodeghero, P., McMillan, C., McBurney, P.W., Bosch, N., D'Mello, S.K.: 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 (2014). DOI 10.1145/2568225.2568247
39. Rojas, J.M., White, T.D., Clegg, B.S., Fraser, G.: 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 (2017). DOI 10.1109/ICSE.2017.68

40. Sedlmair, M., Meyer, M., Munzner, T.: Design study methodology: Reflections from the trenches and the stacks. IEEE Transactions on Visualization and Computer Graphics **18**(12), 2431–2440 (2012). DOI 10.1109/TVCG.2012.213
41. Shneiderman, B.: The New ABCs of Research: Achieving Breakthrough Collaborations, 1st edn. Oxford University Press, Inc., New York, NY, USA (2016)
42. Shull, F.J., Carver, J.C., Vegas, S., Juristo, N.: The role of replications in empirical software engineering. Empirical Software Engineering **13**(2), 211–218 (2008). DOI 10.1007/s10664-008-9060-1
43. Siegmund, J., Siegmund, N., Apel, S.: Views on internal and external validity in empirical software engineering. In: Proceedings of the 37th International Conference on Software Engineering-Volume 1, pp. 9–19. IEEE Press (2015)
44. Sjøberg, D.I., Dybå, T., Anda, B.C., Hannay, J.E.: Building theories in software engineering. In: Guide to advanced empirical software engineering, pp. 312–336. Springer (2008)
45. Stol, K.J., Fitzgerald, B.: Uncovering theories in software engineering. In: 2013 2nd SEMAT Workshop on a General Theory of Software Engineering (GTSE), pp. 5–14 (2013). DOI 10.1109/GTSE.2013.6613863
46. Stol, K.J., Fitzgerald, B.: Theory-oriented software engineering. Science of Computer Programming **101**, 79 – 98 (2015). DOI https://doi.org/10.1016/j.scico.2014.11.010. Towards general theories of software engineering
47. 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
48. Storey, M.A., Engström, E., Höst, M., Runeson, P., Bjarnason, E.: 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 (2017). DOI 10.1109/ESEM.2017.28
49. Tufano, M., Palomba, F., Bavota, G., Oliveto, R., Di Penta, M., De Lucia, A., Poshyvanyk, D.: When and why your code starts to smell bad. In: Proceedings of the 37th International Conference on Software Engineering - Volume 1, ICSE '15, pp. 403–414. IEEE Press, Piscataway, NJ, USA (2015)
50. Wieringa, R.: Design science as nested problem solving. In: Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology, DESRIST '09, pp. 8:1–8:12. ACM, New York, NY, USA (2009). DOI 10.1145/1555619.1555630
51. Wieringa, R., Moralı, A.: Technical action research as a validation method in information systems design science. In: K. Peffers, M. Rothenberger, B. Kuechler (eds.) Design Science Research in Information Systems. Advances in Theory and Practice, pp. 220–238. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
52. Wieringa, R.J.: What Is Design Science? Springer Berlin Heidelberg (2014). DOI 10.1007/978-3-662-43839-8_1
53. Wohlin, C., Aurum, A.: Towards a decision-making structure for selecting a research design in empirical software engineering. Empirical Software Engineering **20**(6), 1427–1455 (2015). DOI 10.1007/s10664-014-9319-7
54. Yan, H., Sui, Y., Chen, S., Xue, J.: Spatio-temporal context reduction: a pointer-analysis-based static approach for detecting use-after-free vulnerabilities. In: M. Chaudron, I. Crnkovic, M. Chechik, M. Harman (eds.) Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018, pp. 327–337. ACM (2018). DOI 10.1145/3180155.3180178