Guiding the Selection of Research Methodology in Industry–Academia Collaboration in Software Engineering

Claes Wohlin [*,a], Per Runeson[b]

[a]Blekinge Institute of Technology, Karlskrona, Sweden
[b]Lund University, Lund, Sweden

Abstract

Background: The literature concerning research methodologies and methods has increased in software engineering in the last decade. However, there is limited guidance on selecting an appropriate research methodology for a given research study or project.
Objective: Based on a selection of research methodologies suitable for software engineering research in collaboration between industry and academia, we present, discuss and compare the methodologies aiming to provide guidance on which research methodology to choose in a given situation to ensure successful industry–academia collaboration in research.
Method: Three research methodologies were chosen for two main reasons. Design Science and Action Research were selected for their usage in software engineering. We also chose a model emanating from software engineering, i.e., the Technology Transfer Model. An overview of each methodology is provided. It is followed by a discussion and an illustration concerning their use in industry–academia collaborative research. The three methodologies are then compared using a set of criteria as a basis for our guidance.
Results: The discussion and comparison of the three research methodologies revealed general similarities and distinct differences. All three research methodologies are easily mapped to the general research process describe–solve–practice, while the main driver behind the formulation of the research

*Corresponding author
Email addresses: claes.wohlin@bth.se (Claes Wohlin *,), per.runeson@cs.lth.se (Per Runeson)

methodologies is different. Thus, we guide in selecting a research methodology given the primary research objective for a given research study or project in collaboration between industry and academia.
Conclusions: We observe that the three research methodologies have different main objectives and differ in some characteristics, although still having a lot in common. We conclude that it is vital to make an informed decision concerning which research methodology to use. The presentation and comparison aim to guide selecting an appropriate research methodology when conducting research in collaboration between industry and academia.

Keywords:
Research methodology, Selecting research methodology, Design Science, Action Research, Technology Transfer Model, Industry–Academia Collaboration

1. Introduction

The awareness of the need to use appropriate research methodologies and methods has increased in software engineering during the last decades. As a socio-technical engineering discipline, it is no surprise that software engineering, particularly in industry-academia collaborative research with empirical evaluations of research solutions, needs to adopt, adapt and be influenced by research methodologies from other disciplines. The influence comes particularly from information systems, engineering disciplines and social, behavioral and management sciences.

Research methodology refers to the research approach, particularly the various types of activities to systematically address the research challenge, which is based on assumptions and justification of the choices made. *Research methodologies* are framed within a *research paradigm*, which is an overarching concept relating to "the set of common beliefs and agreements shared between scientists about how problems should be understood and addressed", according to Kuhn [1]. A vital aspect of a research paradigm is the research methodology, i.e., our approach to acquiring new knowledge. It should be contrasted with *research methods*, which are the means to collect and analyze data, i.e., how the research activities are concretely conducted. Thus, different research methods may be used within a research methodology.

The borderlines between these three concepts are not always clear cut, and it may differ between disciplines and between researchers depending on their view of the world. Our *scope* is on research methodologies suitable for software engineering research in collaboration between industry and academia, particularly when conducting empirical evaluations of solutions developed to address an industrial challenge.

Given our scope, we are targeting methodologies where the intention is to study and improve software engineering practice. We have chosen to focus on three candidate research methodologies, which fulfill the needs of this type of software engineering research. Two of them are brought to software engineering via information systems, i.e., Design Science [2, 3] and Action Research [4, 5], and the third is developed in the software engineering community, the Technology Transfer Model [6, 7]. A fourth one is presented in Appendix A, i.e., the Design Research Methodology, which emanates from mechanical engineering, to complement the other three research methodologies.

The methodologies include different activities, but they are all possible to map to a general engineering research cycle, which we have chosen to summarize in the following three activities: *describe–solve–practice*. The wording is inspired by a discussion by Shaw [8] and further elaborated in Section 2. The three research methodologies are summarized based on their respective background, in particular, based on their use in software engineering.

The article provides novel contributions by presenting, discussing, comparing, and providing guidance concerning selecting a *suitable* research methodology when industry and academia collaborate to derive research solutions that can both be used in industry practice, and being research of high-quality. Thus, the overarching goal is to support successful research in industry-academia collaboration. It should be emphasized that our scope is concerned with research–when–transfer, and not research–then–transfer. The presentations provide overviews of each research methodology. Furthermore, the research methodologies are discussed from the perspective of conducting solution-oriented software engineering research in collaboration between industry and academia. Details concerning the research methodologies are provided, and they are put into a software engineering context, and references to their use in software engineering are provided. Moreover, one example from the literature is summarized to provide more information on how each research methodology may be applied. The three research methodologies are then compared with respect to their main characteristics. Finally, guidance is provided concerning which research methodology to select in a given research situation when conducting research in collaboration between industry and academia. The selection of a research methodology should be firmly based on the research to be undertaken and not based on current knowledge or research tradition.

The remainder of the paper is structured as follows. In Section 2, related work is presented. Our motivation for the research and selection of research methodologies and our research approach is presented in Section 3. In the following three sections, we provide an overview and discuss the three selected research methodologies in the context of conducting software engineering research in collaboration between industry and academia. They are presented as follows: Design Science in Section 4, Action Research in Section 5 and the Technology Transfer Model in Section 6. A comparison of the three research methodologies is provided in Section 7. The conclusions and further work are presented in Section 8.

2. Related work

Shaw denotes software as an engineering discipline, noting that "software engineering shares with classical engineering the need for design techniques to reconcile conflicting constraints and achieve cost-effective results, as well as reliance not only on established scientific knowledge but also on systematically codified observations drawn from experience." [9] In her seminal article "Prospects for an Engineering Discipline of Software" [8], she proposed a cyclic learning process of *new problems*, solved by *ad hoc solutions*; these solutions are shared in a *folklore* style; gradually, knowledge is more systematically *codified*, to become *models and theories*, which may more generally *improve practice* with respect to the original problem.

Her cyclic learning process may be interpreted as two interconnected learning cycles, as illustrated in Figure 1. The first learning cycle is primarily a learning cycle for engineering in practice: new problems—ad hoc solutions—folklore experience from applying the solution in practice. The second learning cycle is more geared towards software engineering research: describe challenge (codify)—develop a general solution (e.g., models and theories)—evaluate and improve practice. The second learning cycle requires a research methodology with suitable activities to deliver research results useful in practice.

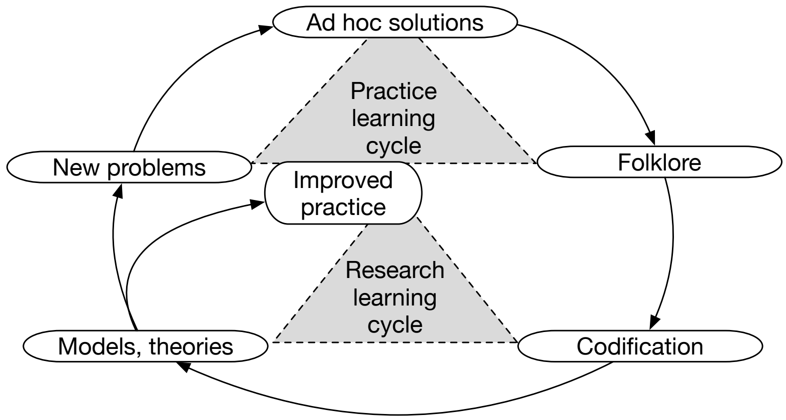

Figure 1: Shaw's learning cycle [8], augmented with practice and research learning cycles.

Building on the terminology of Shaw, software engineering research may be viewed as a cyclic research process, which begins and ends in relevant practice and theory. We have chosen to describe the three activities in the cyclic research process as *describe–solve–practice*, where practice refers to the evaluation and practical use of the solution. The cyclic research process for software engineering is illustrated in Figure 2. Wieringa proposed a cyclic process with three activities along the lines suggested here when using *investigate–solve–validate* [10]. In particular, we chose to use the verb "practice" to highlight the collaboration between industry and academia, with the objective to put the research solution into use in an industrial context.

Similar cycles exist for research in other disciplines. For example, Agnew and Pyke [11] suggest *observe–think–test* in behavioral and social science. For research in information systems, Venable [12] suggests adding a fourth activity in the middle called *theory building*, which is connected to the other three activities in Figure 2. This activity aims to capture the generalized learning from the problem–solving activities represented by the three activities.

Different aspects concerning industry–academia collaboration in software engineering are summarized in two systematic literature reviews by Garousi et al. [13] and Brings et al. [14], while Wohlin et al. [15] presented success factors for industry–academia collaboration. Mikkonen et al. [16] highlighted the need for close and continuous collaboration between industry and academia. Furthermore, they stressed the need for technology pull by industry instead of technology push by academia, resulting in industry–academia co-creation. The collaborative challenge is not new. As early as 1997, Beck-

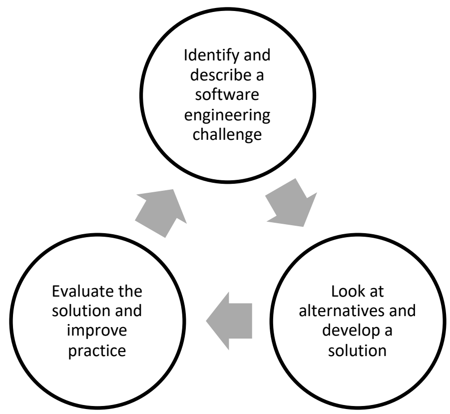

Figure 2: The *describe–solve–practice* research cycle for software engineering research.

man et al. [17] stressed the need to close the gap between industry and academia, and Sandberg et al. [18] discussed what they refer to as agile collaborative research. The need for industry–academia collaboration is also part of the reasoning by Shaw [8], as discussed above, when she highlights "evaluate and improve practice". However, we have not identified any article explicitly addressing the challenge of selecting research methodology when conducting research in collaboration between industry and academia.

Stol and Fitzgerald [19] highlighted the natural setting (field studies) as one of four types of software engineering research settings. Storey et al. [20] use the same model when classifying articles. They argue that we need to "improve the relevance of our research for human stakeholders". Their argument builds on an analysis of 151 articles published in the International Conference on Software Engineering and the Journal of Empirical Software Engineering. Out of the 151 articles, only ten articles are classified into the field studies category. Thus, more studies are needed in the natural setting, which requires improving the collaboration between industry and academia.

We now turn to empirical evaluations in software engineering, which have been around for more than 50 years [21]. In industry–academia collaboration, empirical methods such as surveys, interviews and case studies are essential. Chapters on these methods are provided in edited volumes such as Shull et al. [22] and Felderer and Travassos [21]. Furthermore, books on empirical methods have been presented, for example, the book on case studies in software engineering [23].

Based on the need to choose research methods for evaluating and assessing software engineering solutions, authors have provided guidance in selecting appropriate research methods. For example, Easterbrook et al. presented guidelines for choosing empirical methods [24], and Wohlin and Aurum [25] provided a decision-making structure for selecting the research design in empirical software engineering. Stol and Fitzgerald [19] take it one step further by applying a framework from social sciences to discuss eight research strategies in software engineering. Their objective is to trade "the level of obtrusiveness of the research, and generalizability of research findings".

In recent years, the interest not only for research methods but also for research methodologies has increased, and books on design science [3] and action research [5] in a software engineering context have been published. However, the support for selecting an appropriate research methodology in a given situation is limited. The objective here is to provide guidance concerning selecting among the three research methodologies presented in Sections 4–6 and further compared in Section 7.

3. Approaching the research methodologies

3.1. Motivation for research and the selection of methodologies

Our analysis of related work shows 1) the lack of guidance for methodology selection and 2) the need for more industry–academia collaboration to improve the research relevance. Furthermore, the importance of context is also argued by Briand et al. [26] when they put forward the need for context-driven software engineering research. Basili et al. [27] continued the discussion along the same lines by stressing the importance of context. Thus, the collaboration between industry and academia is by many viewed as essential. However, the collaboration between industry and academia in research is by no means easy given the different timelines and objectives with the collaboration as discussed by Runeson et al. [28].

One challenge in the collaboration between industry and academia is identifying a collaborative research approach that benefits both parties, and provide credible evidence [29]. This implies that support is needed to select an appropriate research methodology to increase the likelihood of successful collaboration. We have chosen three research methodologies, which by no means is an exhaustive selection. However, it is three strong contenders when it comes to selecting an appropriate research methodology when collaborating between industry and academia.

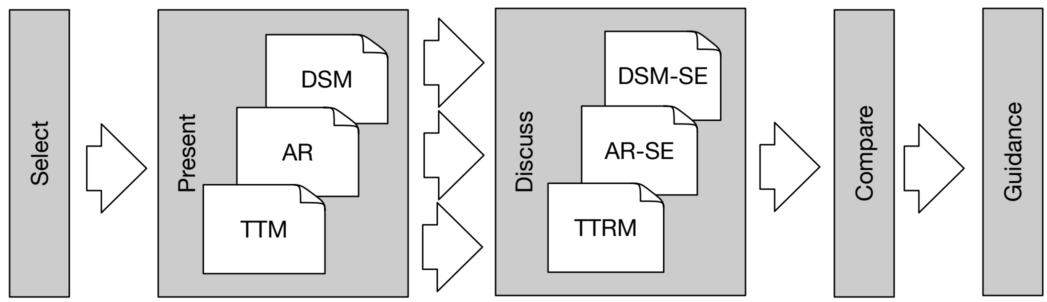

Figure 3: Overview of steps in our research.

We have focused on solution-oriented research methodologies, i.e. when we set out to provide useful research solution to industrial challenges. The motivation for selecting the three methodologies is as follows. For two of them, i.e., Design Science Methodology (DSM) and Action Research (AR), we have seen books, articles and book chapters published in the context of software engineering. Thus, these two methodologies were natural candidates to include when discussing research methodologies in software engineering. As a third research methodology, the Technology Transfer Model (TTM) was included given that it emanates from a software engineering context, and it has been used by several researchers in software engineering, as further discussed in Section 6.2. Due to the solution–oriented scope, we have not included research methodologies focusing more on understanding and describing software engineering practices such as Grounded Theory discussed by, e.g. Stol et al. [30] and Ethnography presented by, e.g. Sharp et al. [31].

3.2. Research approach

Our research approach is a theoretical analysis of multiple research methodologies through thematic synthesis [32] in the context of our extensive experience from industrial collaboration and empirical research in software engineering. The overall objective is to present, discuss and compare research methodologies to provide guidance concerning the selection between them for software engineering research when conducted as a collaboration between industry and academia. The research work is iterated in several cycles. For each cycle, tasks were divided among the authors and next reviewed by the other author. The major steps in our research method are presented in Figure 3 and explored below.

Based on the above objective, the following research questions were formulated:

- RQ1: What are the characteristics of the different research methodologies?
- RQ2: What are the main similarities and differences between the research methodologies?
- RQ3: How do we select a suitable research methodology in a specific situation?

To start addressing the research questions, we studied the literature on the three methodologies and summarized them primarily as *presented* in their fields of origin. The overviews in Sections 4.1, 5.1 and 6.1, include the origin of the methodologies and a description of their core characteristics. Depending on the heterogeneity of the respective methodology, they are presented at different levels of detail. During the information collection, we also gathered aspects to compare the methodologies.

DSM and AR are then *discussed* from an industry–academia collaboration perspective in software engineering. We do not provide such a discussion for TTM, which emerged in a software engineering context. The outcome concerning DSM and AR, presented in Sections 4.2 and 5.2 respectively, varies depending on to what extent each methodology has been applied in software engineering. For methodologies existing in multiple variants, we selected one as a reference that was synthesized from various sources [2] or established as the practice [33].

Furthermore, TTM is not primarily put forward as a research methodology, although it has been used as such. Thus, we have chosen to reformulate the technology transfer model to gear it more towards being a research methodology than originally presented [6]. It is here called the Technology Transfer Research Methodology (TTRM), as shown in Figure 3.

For each research methodology, some references illustrating its use in practice are provided. One of the identified examples is described in some further details to illustrate each methodology, provided in Sections 4.3, 5.3 and 6.2. The selection of illustrations was based on the following criteria:

- Use different research methods within the use of the research methodology,

- Other researchers have conducted the study reported than the authors of this article,

- If possible, illustrate the comparison aspects, which are listed in the next section.

Based on our analysis of each methodology, we then *compared* them based on several aspects. The comparison is presented in Section 7, which also provides *guidance* for the selection of a research methodology in software engineering.

3.3. Aspects of comparison

In the comparison, we start with comparing the primary objectives of the research methodologies. Next, the following aspects are compared:

- Origin – This refers to the background of the respective research methodology, i.e. from which research domain the methodology originates.

- Outcome – Different research methodologies may have different main and secondary outcomes.

- Roles – Research methodologies may include different roles, particularly which roles practitioners and researchers take.

- Driver of literature search – Related work is an essential part of research in general, but different research methodologies may approach it differently, and in particular, the motivation may differ.

- Learning – The methodologies may have different objectives with respect to generalized and case specific learning, and hence different primary expected lessons learned.

4. Design Science Methodologies

4.1. Overview

Design science may denote a *paradigm* or *methodology*. We do not take a stand in this sometimes infected debate on whether it is a paradigm or methodology [34, 35] but discuss the views as being possible to coexist. As mentioned in Section 1, paradigm refers to “the set of common beliefs and agreements shared between scientist about how problems should be understood and addressed", according to Kuhn [1]. As a paradigm, design science embraces research on designed phenomena in contrast to research explaining naturally occurring phenomena, or formalizing philosophical or logical systems. For example, information systems research is often conducted under the design science paradigm [36], and so is management science [37], while physics typically is explanatory, and mathematics is a dominantly formal branch of science. Design science as a paradigm for software engineering is explored by Runeson et al. [38] and Engström et al. [39].

As defined in Section 1, we refer to a research methodology as the approach to research, particularly the various types of activities to systematically address the research challenge, which is based on assumptions and justification of the choices made. DSM, as a research methodology, provides prescriptive guidance and frameworks. DSM is at a more concrete level than the Design Science Paradigm and depicts one of many potential methodologies within the paradigm. However, a DSM is flexible enough to allow many different research methods for data collection and analysis to be utilized within the DSM frame.

Design science emerged from the desire to conduct research on artificial, or designed constructs, in contrast to naturally occurring phenomena [40]. Simon, a 1978 Nobel Prize laureate in Economics, defined engineering as the historical roots of design science. However, he embraces all kinds of design, for example, architecture, business, education, law, and medicine [40]. Consequently, the primary outcome of the research is the artifact emerging from the design, accompanied by knowledge about the design process.

Several instances of DSM are proposed and used in different fields of research, e.g. by Hevner et al. [36] and Peffers et al. [41] in the field of information systems, and Wieringa, bridging information systems and software engineering [3]. Offermann et al. [2] compare four existing "design science research processes" and propose a fifth one, which we discuss in Section 4.2.

All of these instances encapsulate three generic activities:

1. Problem identification or conceptualization
   The problem under study has to be understood in its context. This is not only a description or enumeration of problems but an in-depth understanding of the constituents of the problem – its concepts. Taxonomies or model proposals may emerge from problem conceptualization, or existing theories are used as concepts in the problem identifi-

cation.

2. Solution or artifact design and implementation
   The envisioned solution emerges from the problem identification, supported by existing knowledge of the field. Solutions may be instances of earlier known general solutions or developed specifically for the particular problem. Design knowledge is used and produced in the solution design process, often referred to as an artifact.
3. Evaluation or validation
   This activity aims to assess to what degree the solution solves the problem. If the solution is instantiated in a new type of context, it also extends the scope of the solution's validity.

DSMs stress that the research is conducted in a context of practice and generates contributions to practice. Multiple case studies are brought forward as the typical research method for design science research [37], both for the problem conceptualization and the evaluation. Some scholars label the outcomes *artifacts*, although there is no unified taxonomy for what an artifact is. Offermann et al. reviewed design science literature and synthesized artifact types from 106 papers, identifying the following categories of artifacts: system designs, methods, languages, algorithms, guidelines, requirements, patterns, and metrics [42]. Van Aken defines *"technological rules"* as the primary contributions from design science, which means "field-tested and grounded" exemplars of how a problem can be solved [37], typically in the form: *To achieve <Effect> in <Context> apply <Intervention>.*

Furthermore, DSMs stress the research to i) be built on existing knowledge or theory, and ii) create/generate/synthesize design knowledge, in more or less generalized form, about the area of study. Venable focuses on the practical utility in design science by defining: "Theory embodies statements of knowledge...in a form that has both use in the practical world...and in the theoretical world" [12]. This duality is presented as rigor versus relevance cycles by Hevner [36] and is put forward as an "act of producing knowledge by designing useful things" by Wieringa [10]. Wieringa further distinguishes between knowledge problems and practical problems to address these dual goals, which are also present in Shaw's practice and research learning cycles, as illustrated in Figure 1 [8].

4.2. DSM for industry–academia collaboration in software engineering

As an example of DSM, we use the research process proposed by Offermann et al. [2] since it is synthesized from several other methodologies and

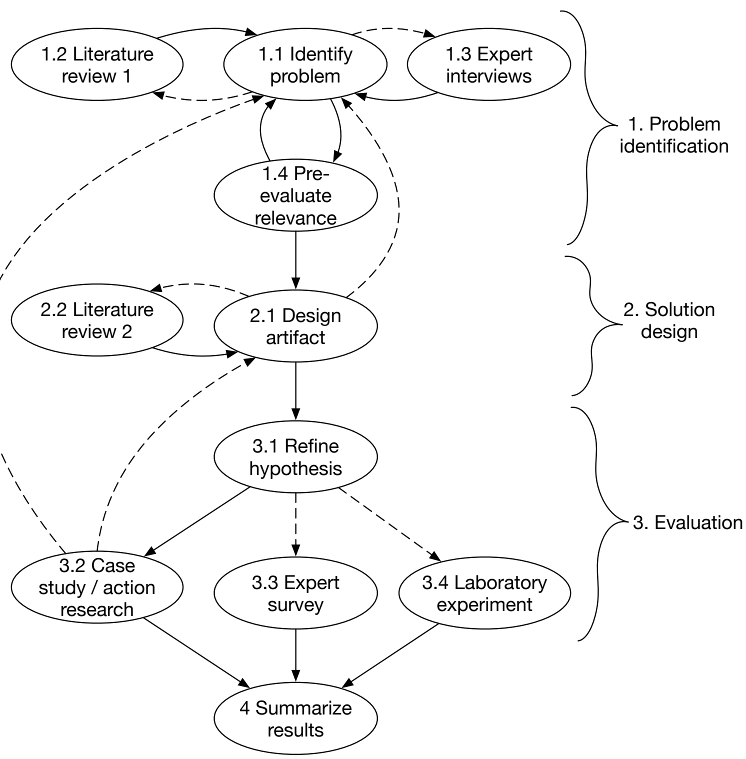

Figure 4: Research process for Design Science Methodology research, adapted from a proposal by Offermann et al. [2]. Ovals represent activities which are grouped into three main phases. Dashed lines indicate optional paths.

maps well to the general problem-solving model. They present a primarily linear process model, with some iterations, see Figure 4.

The model contains 11 activities, which are grouped into three main phases: problem identification, solution design, and evaluation. These phases map perfectly onto the general engineering research cycle in Figure 2.

The first phase concerns *problem identification* and contains activities to identify a relevant research problem within the domain of study. The problems shall be rooted in the literature and practice, implying some form of industry–academia collaboration to ensure both. To support the problem identification, a *first literature study* may be conducted of scientific publications and practitioner reports. The search aims to find knowledge, both about the problem and potential solutions to the problem. As proposed in software engineering, the search for practitioner reports implies that gray literature also is taken into account, as discussed by Garousi et al. [43]. However, they do not claim any general requirement on the rigor of the literature review, in contrast to software engineering research, where systematic literature reviews and mapping studies gradually have evolved as a norm [44].

The phase may also encompass *expert interviews* to help understand the identified problem and assess its relevance. Interviews could take place one by one or in workshops. This practice is well established in software engineering as focus groups [45]. Offermann et al. stress that the problem should be of interest to more than one organization. If not, it should be generalized to make it relevant for more actors [2]. It is worth noting that they do *not* propose case studies to identify the research problem, in contrast to software engineering guidelines, where exploratory case studies are proposed to be conducted for "generating ideas and hypotheses for new research" [46].

Next, a *pre-evaluation relevance* assessment is conducted. This includes stating a research hypothesis "in the form of a utility theory" [2]. Venable [12] defines that a utility theory should express a hypothesized connection between the solution and the problem space, i.e. what is the problem and how is the proposed solution expected to address that. Offermann et al. [2] propose the hypothesis be expressed in a specific format: "if a solution to the problem is applied, some observed aspects will be changed in a way which ultimately helps the entities". It resembles the technological rule brought forward by van Aken [37]. In software engineering research, it is more common to express the utility theory in terms of research questions, for which there also exist empirically derived guidelines in information systems [47].

The second phase, *solution design*, is a creative engineering process. It is not much prescribed, but for the guidance about taking existing solutions and state-of-the-art into account. For this purpose, a *second literature review* may be conducted, now focusing on scientific literature searching for solutions. During this phase, a need for better understanding or a revision of the problem may be identified, thus iterating back to the problem identification phase. Other design science methodologies stress that alternative solutions should be considered. For example, Johannesson and Perjons [48] propose *divergent thinking*, meaning generating multiple, alternative ideas or solutions to address a problem, and *convergent thinking*, meaning evaluation of and selection from alternative ideas generated. The primary outcome of this phase is, however, the *artifact*. In software engineering research, typical artifacts are tools, models or techniques, in which design knowledge is embedded [39].

The third phase, *evaluation*, starts by refining the research hypothesis. The overall research hypothesis may be too general or comprehensive, to be feasible for evaluation. Therefore, the hypothesis is refined to be more specific or limited in scope. The primary evaluation method is *case study* or *action research*, but may also embody *expert surveys*, or *laboratory experiments*. In this context, action research is seen as a research method, not a methodology. Also, here, the process may iterate back for deeper problem understanding or improved solution design.

The evaluation is at the core of empirical research in software engineering, where the authors of this article have contributed to method guidelines for experiments [49], case studies [23] and systematic literature reviews [44, 50, 51], and Staron recently contributed with an action research guidebook [5]. Which approach to use depends on the goal of the evaluation, whether generalizability, precision in measurements, or realism is prioritized. Stol and Fitzgerald provide guidance in this respect through the ABC framework [19]. They have further recently integrated the ABC framework with the design science perspective [52].

In the final step, the *results are summarized* and published in feasible formats. Intermediate results may also be published, and thereby early feedback on the results can be gained.

Engström et al. conducted an analysis of 38 distinguished papers at the ICSE conferences 2014–2018 from a design science perspective [39]. They conclude that most papers can be expressed from a design science perspective: solution–oriented, aiming to provide design knowledge, although it is less clear which practical problem they address. They observe that the solution design process is often implicit, although defined in terms of the previous and updated state of a technology or process under improvement. Only 13 out of the 38 papers reported a complete problem–solution pair. However, a design science research project can be published partially in several papers. Eight papers focused on describing the problem, seven on solution design and seven on solution validation. Still, the outcome of the analysis is that this part of the software engineering community would benefit from being better anchored in practice, again stressing the need for closer industry–academia collaboration.

In summary, the design science methodology, as formulated by Offermann et al. [2], aligns well with software engineering research and the general research cycle. Software engineering research may put even more emphasis on the empirical methods in problem identification, and the notion of an artifact may differ from the information systems concepts.

4.3. Illustration of DSM

DSM is used to some extent in software engineering, although the design science approach is often implicit, as mentioned above [39]. However, there are studies explicitly following DSM, foremost referring to Hevner [36] or Wieringa [10, 53, 3] for their methodology. Examples include design and evaluation of an artifact-based requirements engineering model [54], specification of confidentiality requirements [55], design and evaluation of a lightweight analysis method to find root causes for defects [56], and a method to improve the alignment between test and requirements, by reducing communication gaps [57]. Furthermore, DSM is used in research where the design of the software itself is the resulting artifact [58] and to the creation of software engineering research tools to mine GitHub data [59].

To illustrate DSM, we selected the work by Bjarnason et al. [57], where they designed and evaluated an artifact, namely Gap Finder, which is a maturity and improvement model. Expressed as a technological rule (see Section 4.1 and [37]) they recommend using "Gap Finder for assessing and identifying suitable improvements to the alignment of requirements and testing within a software development project" [57].

Its theoretical underpinning comes from Bjarnason et al.'s theory of distances in software engineering [60], which in turn was based on literature reviews and empirical observations of practice in five companies. The theory of distances explains how certain software engineering practices improve the communication within a project by impacting distances between people, activities and artifacts.

They proceeded the work using DSM to design a practical method for applying the theory in practice. Figure 4 is used below as a frame of reference in the illustration. A case organization was studied to describe the problem of communication gaps between requirements and testing and to validate its practical relevance (1. Problem identification). Semi-structured interviews, document studies, and ethnographic observations were used to identify problems. Thus, they used more thorough empirical methods for the problem identification, compared to the DSM process by Offermann et al. [2].

Gap Finder was then iteratively designed to guide the systematic assessment of organizations, which is a key characteristic of DSM. The research team designed (2.1 Design artifact) the initial version based on the underpinning theory (2.2 Literature review 2) and the knowledge gained from the studied case in the first round of interviews, document studies, and observations. As a first evaluation, the scope was reduced (3.1 Refine hypothesis) and it was applied to one sprint for one development team to help the researchers assess its feasibility and evolve the Gap Finder (3.2 Case study). They gained new insights from the case, improving the design for the specific needs (2.1 Design artifact, 2nd iteration) while maintaining its generality.

The resulting method contains an assessment process with workshop and measurement guidelines, outcome presented in radar diagrams, and recommendations for improved practice.

The method was then evaluated in the case context by applying the assessment process of the Gap Finder (3.2 Case study, 2nd iteration). In addition to the data collected in the method, a focus group session and a survey were organized to collect specific validation feedback (3.3 Expert survey). The validation concerned the evaluation method as such, as well as the proposed practices to reduce gaps between requirements and testing.

In summary, this DSM study revolves around the derivation of knowledge about reducing requirements–test communication gaps, embedded in an artifact, the Gap Finder method. Both problems and solutions were derived from existing empirically based knowledge, while the change process is left to the organization to continue with support from the Gap Finder method.

5. Action Research

5.1. Overview

Action Research (AR) has its roots in social science and was developed to change a social system while doing research. It emerged as a reaction to research only creating knowledge without setting it into action, thus focusing on the researchers being involved in a change process. Lewin coined the term in the 1940s [61], and it became gradually more defined and prescriptive, particularly by Susman and Evered [62], introducing a cyclical process of five phases:

1. Diagnosing – identifying or defining a practical problem to be addressed in collaboration between researchers and practitioners. The researcher should confirm the problems identified by the practitioners and also determine the root causes of the problems.
2. Action planning – considering alternative approaches to solve the problem. The diagnosing phase informs the action planning towards potential solutions to the problem under study.

3. Action taking – setting the planned actions into practice. The change process may need support from actors in various roles of the organization, for example, a project champion who helps initiate changes.
4. Evaluating – studying the consequences of an action. Data collection should be performed before, during and after the action taken to ensure that enough information is gathered to assess the goals of the actions and the mechanisms leading to the change.
5. Specifying/learning – identifying general findings in relation to the problem and actions under study. The primary focus is set with regards to the current project, and decisions related to the exit of the researcher or initiation of a new cycle is taken.

This cyclical process is widely adopted in the social sciences and constitutes one of the five principles for Canonical Action Research (CAR) [33]. The other four principles are about researcher–client agreement, theory, change through action, and learning through reflection. Specifically, the principle of theory focuses on using theory to guide research rather than generating or validating theory as an action research goal. However, AR comes in many forms. In 1948, AR pioneers Chein et al. [63] identified four varieties of AR, depending on the degree and type of interaction between researchers and the community under study. Baskerville and Wood-Harper provide an overview of the evolution of AR in social science and information systems, identifying ten distinguishable forms of AR in the information systems literature [4].

Action Research was first introduced into information systems as a research methodology by Wood-Harper in 1985 [64] and then into software engineering around the turn of the millennium, according to a literature review by Santos and Travassos [65]. Guidelines for action research in software engineering are published by Santos and Travassos [65], Wieringa [66], and Staron [5].

AR methodologies, in general, have been criticized for lacking rigor while generally producing relevant results [33]. Davison does not refute the criticism but argues against rigor and relevance in AR having an inverse relationship. “[T]hese two attributes need not be mutually exclusive, although they can be hard to achieve in a single CAR project.” [33]

The relation between action research and design science is another point of debate in the information systems community. While Järvinen argues that “Action research is similar to design science” [67], Iilari and Venable claim AR and DS are “Seemingly similar but decisively dissimilar” [35]. Baskerville states that “Design science is not action research” [34]. We do not want to enroll in these methodological wars but conclude that the differences depend on whether they compare AR with DSM or design science as a paradigm. In this paper, we choose, like Wohlin and Aurum [25], to treat both action research and design science as methodologies to get comparable entities in our analysis.

5.2. AR for industry–academia collaboration in software engineering

Our discussion concerning AR takes its starting point from CAR (i.e. Canonical Action Research), as defined by Susman and Evered [62] and later refined by Davison [33], see Section 5.1. We further analyze Staron’s proposed instantiation of AR for software engineering [5]. The five phases of CAR map to the three activities in the general research cycle illustrated in Figure 2:

- describe – diagnosing the problem
- solve – action planning and action-taking
- practice – evaluating and specifying/learning

CAR stresses the cyclic procedure and that there may be more fine-grained iterations, e.g. if the proposed actions have to be revisited. Furthermore, Davison refers to early proposals of a spiral model of multiple cycles from Kemmis and McTaggart [68], where each cycle focuses closer on the organizational problem under study. A similar approach was used in software engineering by Andersson and Runeson [69], inspired by Boehm’s spiral process model for software engineering [70], see Figure 5. Their research endeavor on software quality monitoring included seven cycles, starting with a modeling activity, followed by two exploratory cycles, which were confirmed in the fourth cycle. The three last cycles were explanatory and aimed to develop quality prediction models. All cycles were performed in industry–academia collaboration, although tasks were divided between the groups. Goals and scope for each iteration were set jointly, researchers collected and analyzed data, while the practitioners were responsible for the change action and working procedures in the organization.

The social context, and the change of the conditions of the social context, is in focus for AR. Baskerville defines as a key assumption for AR that the “social setting cannot be reduced for study” [71], implying that the phenomenon under study loses key characteristics if taken out of its context.

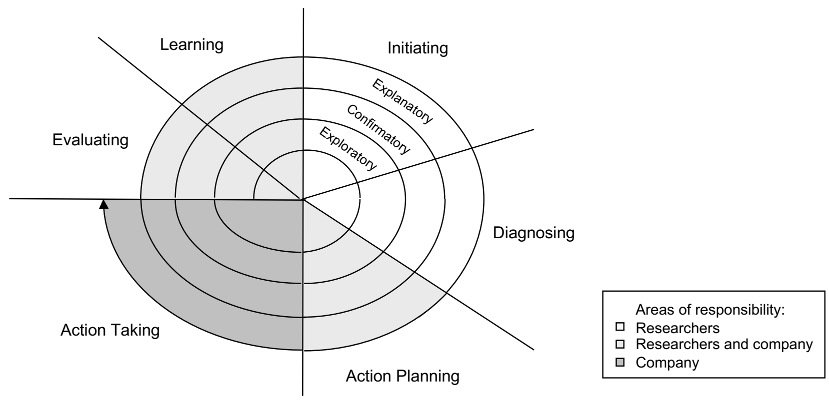

Figure 5: Spiral research model adapted from Andersson and Runeson [69] to the CAR terminology.

This fits well with particular aspects of software engineering research, where complex interactions in a socio-technical system are at the core of its research challenges. For example, Runeson et al. define a software engineering case study to be "... an empirical enquiry ... in its real-life context, especially when the boundary between phenomenon and context cannot be clearly specified." [23, p. 12] Furthermore, Wohlin et al. put forward a general theory for software engineering based on balancing human, social and organizational capitals [72].

Santos and Travassos surveyed the software engineering literature 1993–2010 and found an increasing trend of the use of AR, although at a low level. In total, 22 studies were found in nine high-quality software engineering journals and four conferences [65]. We have not found any later systematic literature review on AR in software engineering. However, database searches indicate more AR studies published in software engineering after 2010, although still at a low pace.

There are several proposals for the adaptation of AR to software engineering. Santos and Travassos [65] elaborated on the interplay between the change action and the theoretical learning, arguing that "the Action Research methodology with its dual objective of improving organizational problems and generating scientific knowledge leads to a 'win–win' scenario for both professionals (organization) and researchers."

Wieringa defined the notion of Technical Action Research (TAR), using DSM guidelines to design an artifact and AR practices to scale it up to

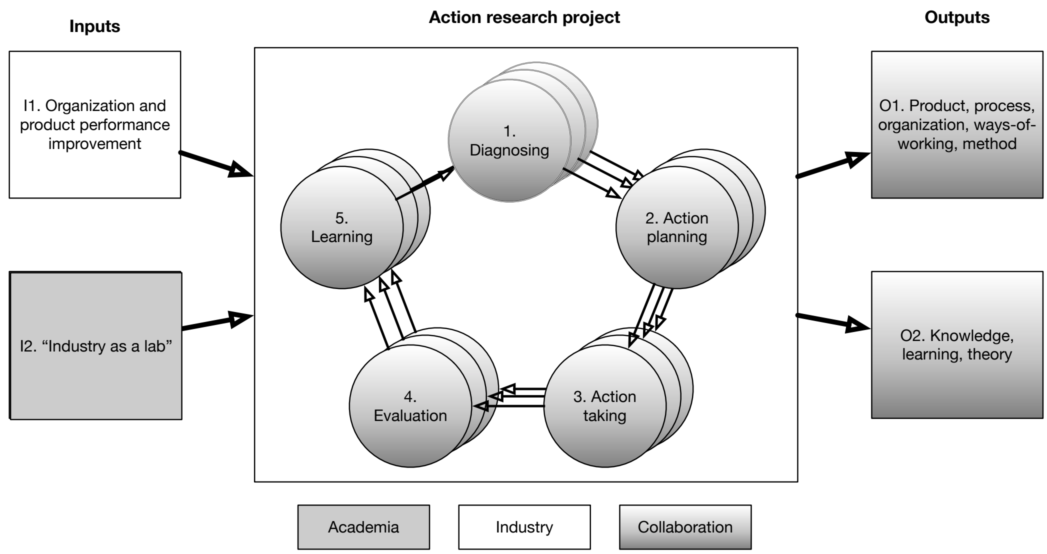

Figure 6: Overview of Staron's AR model for software engineering research [5]

practice and validate it [66]. Petersen et al. proposed AR as an industry–academia collaboration model [73]. Garousi et al. recently proposed AR as a feasible approach to improve the relevance of software engineering research, based on their analysis of peer–reviewed and gray literature on academic and practical relevance [74]. Thus, there are several applications of AR to software engineering, indicating its feasibility for software engineering.

Staron, who has published several papers on AR research in software engineering, recently published comprehensive practical guidelines for AR in software engineering, including the process of industry–academia collaboration at large [5]. Staron's guidelines extend CAR by contextualizing the five phases for software engineering companies and research projects through experience-based recommendations. An overview of the AR model is presented in Figure 6. Furthermore, he defines three key team roles in the research collaboration, namely:

- The action team is responsible for planning, executing and evaluating the research.
- The reference group is responsible for the advice and feedback to the action team.
- The management team is responsible for managing and governing the

project and its institutionalization of changes.

The action team comprises both researchers and practitioners, while the reference group and management teams consist of company representatives only.

Action research methodology has several characteristics that fit software engineering research, particularly for collaboration–intensive industry–academia projects, focusing on the change of industrial practices. The cyclic approach, the problem–oriented focus, and the empirical evaluations in real–life contexts resonate well with successful collaboration projects, exemplified by Carver and Prikladnicki [75]. They also stress the importance of feasible roles in the collaboration, which also is a key asset of CAR [33].

Wohlin et al. confirmed the importance of roles in a survey with researchers and practitioners [15]. Both groups ranked the role of a champion on–site as the top 1 or top 2 factor for a successful industry–academia collaboration. Still, there are different views on the balance between the researcher and the practitioner in AR. Davison states that "[r]esearchers typically guide the overall process, but their scope of responsibility on content issues is a common topic of negotiation." [33] Staron [5, p.22] stresses that the action team comprises both researchers and practitioners without detailing who is doing what within the team. They complement each other based on their different knowledge and perspectives. Decision making concerning change proposals involves an interplay with the management team. Andersson and Runeson [69] make the separation between researchers and practitioners clearer by assigning responsibility for action taking to the company, while the researchers are responsible for the diagnosis, see Figure 5.

Software engineering is a socio-technical field, and as AR emerges from the social sciences, it strengthens the social side of the research and opens up towards qualitative research methods [76]. Software engineering challenges is often an interplay between technical and organizational or human aspects. Notably, Staron defines AR as a "quantitative methodology", arguing that quantification "provide[s] the possibility to reduce the bias of subjective observations and provide quantitative evidence" [5, p.19]. In contrast, the software engineering community gradually has evolved towards taking on social sciences' qualitative methodologies (e.g. Seaman [76] and Runeson et al. [23]) and learned to address the validity of such studies too.

Action research has been criticized for being more action than research [33]. On the contrary, software engineering research has been criticized for being more research than action. Particularly, Briand et al. [26] argue for problem–

driven research in collaboration with practitioners conducted in specific contexts. This is indeed a trade–off that has to be handled properly. Conducting research in complex contexts add to the relevance of the research. On the other hand, there is a risk that the research becomes "advocacy research" in the sense that researchers advocate for their proposed solutions and look only for signs of confirmation of their hypothesis. However, this can be handled by adhering to proper research and validation procedures for both qualitative and quantitative research.

As the primary goal of AR is to support change in a specific context, generalization is given less priority. However, Davison et al. strongly argue for the role of theory in CAR to guide the research and as an output that helps to communicate generalized knowledge [33, p.74]. In software engineering, the use of theory is not very prevalent [77, 78]. Staron, therefore, extends the notion of theory: "The theory, in this context, is the description of the phenomena that need to be studied, theoretical relationships between elements of that phenomenon, and the rationale behind them." [5, p.38] This is what may be called a hypothesis or a model in other contexts, but still, the AR model embraces the scientific knowledge–building process.

In summary, AR is a feasible research methodology for software engineering research in industry–academia collaboration, focusing on the change of practice. It has limitations with risks for focusing too much on action, but it is a sound counteract against too much research in the lab for the software engineering community. Theorizing is embedded in the methodology but rather a means or a by–product than a primary goal.

5.3. Illustration of AR

Action research studies exist in software engineering, although they tend to be more inspired by the AR principles rather than fully adhering to AR methodological guidelines. However, such studies are in emergence, as mentioned above, in relation to Staron's guidelines [5]. Staron and co-authors recently published two AR studies, where two companies worked together on improving tool support for selecting code fragments for review [79] and identifying violations of coding guidelines [80], respectively. Choras et al. also used software engineering tools as a vehicle to improve practice [81], and Razavian and Lago developed a strategy for data migration [82], based on Wieringa's TAR guidelines [66]. These TAR studies focus more on the design and less on diagnosis than the original CAR principles [62]. This is, in fact, the aim of TAR with its dual engineering and research cycles, focusing on action and generalized design knowledge, respectively. To illustrate AR, as such, we select Ananjeva et al.'s study, aiming to integrate user experience (UX) work with agile software development, as our illustrative example [83], referring to artifacts and activities in Figure 6.

The AR study is conducted jointly (Collaboration) between a small software–as–a–service company (Industry), an on–site observer (Master thesis student), and researchers with software engineering and UX competence (Academia) over 12 months. The company observed challenges integrating their UX work in their agile development process, performed by two separate teams (Input I1). The problems were explored in depth through five months of on–site observations, interleaved with 32 recorded interviews and ad hoc conversations with team members (1. Diagnosing). The primary concern identified, was in short, that the user stories were too lengthy to match with agile principles.

The first proposed intervention (2. Action planning) was to write more concise user stories and complement them with face–to–face communication. This intervention proposal was discussed at length and criticized by one of the managers – according to the researchers, wanting "verbose user stories as a shield to protect herself and the stories from [...] criticism". Thus, as a second intervention, a workshop was organized to help the two teams discuss and resolve the latent conflict about the user stories. They came up with a compromise, which they decided to implement (3. Action taking).

In a follow-up workshop two months later, the managers of the two teams confirmed that they were in the process of eliminating the verbose user stories, but they had not yet succeeded (4. Evaluation). The researchers also reflect on the lessons learned from the case, demonstrating the interplay between the organization, its culture, and the development processes (5. Learning).

In summary, the AR study focuses on the change in the organization (Output O1). It is an iterative and highly intertwined endeavor between researchers and practitioners. This example demonstrates how a technical change proposal triggers a social or organizational conflict, which has to be addressed in conjunction with the change. The study is well anchored in existing theory, while the outcome is rather lessons learned than generalized knowledge (Output O2).

6. Technology Transfer Model

6.1. Overview

Gorschek et al. [84] formulated a model for technology transfer with several activities for conducting industrially relevant research and then transfer the research outcomes to practice. It includes working very closely between practitioners and researchers throughout the process to create trust and commitment from the practitioners concerning the solutions developed as part of the research. Hence, the model is developed with a strong focus on academia–industry collaboration, where the problem being researched emanates from an industrial challenge. The solution developed through research goes through several validation steps to ensure that it may be put into practice with low risk.

The model highlights three different roles: *practitioner, champion* and *researcher*. Practitioner refers to stakeholders in the industry, potentially with varying roles in the company. In comparison, the champion is a driver of the research collaboration from the industry side. The champion is the primary contact person and is supposed to help with the right contacts within the company and to ensure a broader company commitment. Commitment is needed from both adequate technical staff and appropriate management levels. In the different activities, it is essential to have comprehensive coverage of roles and stakeholders to ensure a smooth transfer of the solution once it put into practice.

The model is not formulated as a research methodology as such. However, the activities in the model describe the process of conducting research in close collaboration between industry and academia. The model is illustrated in Figure 7, and the activities may be summarized as follows:

1. Industrial challenge
   The research should be based on industry needs identified in a dialogue between industrial partners and the researchers. The identification includes process assessment and observation activities. Having researchers present at collaborative partner sites is highlighted as essential to ensure good personal contacts and to build trust on an individual level.
2. Problem statement and state-of-the-art
   The industrial challenge from item 1 is formulated in terms of research and, in particular, research questions. Furthermore, the literature is studied to capture relevant research concerning the industrial challenge.

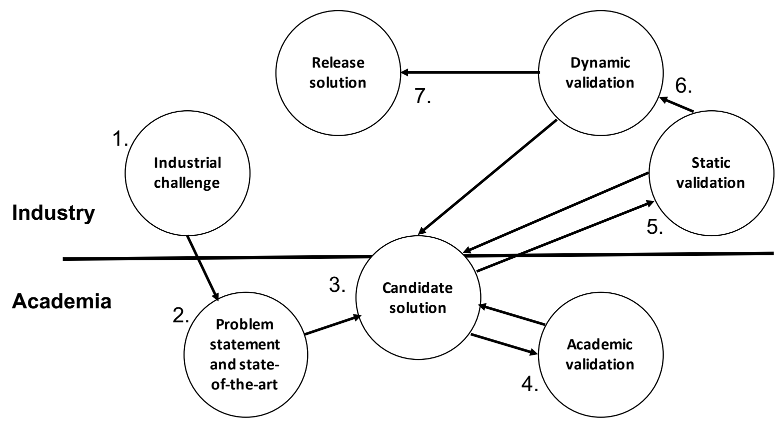

Figure 7: The technology transfer model adapted from Gorschek et al. [84].

3. Candidate solution
   In close dialogue with the industrial partner(s), and, in particular, the champion(s), a solution is developed. The solution should be based on relevant existing research, novel research ideas as part of the collaboration and the industrial context that the solution should fit into.
4. Academic validation
   To minimize risk before transferring a solution to the industry, it is preferably validated in an academic setting. This may be done, for example, through experimentation with human subjects [49] or simulation [85].
5. Static validation
   Once no further improvements are identified in the academic setting, a static validation is conducted. The static validation is done through seminars and discussions with the key stakeholders to anchor the proposed solution. Based on potential feedback from the static validation, the solution is refined. Depending on the changes to the solution, there may be a need for more than one round of static validation.
6. Dynamic validation
   Dynamic validation means running a pilot in a suitable situation. The pilot should be as representative as possible of the regular context in which the solution is expected to be used. It should help ensure that the solution fits well with the current practices and the potential roles and responsibilities put forward by the solution. Based on the outcome, the solution may be updated or fine-tuned, and it then has to be decided

whether further pilots are needed, or the solution is ready for broader usage within the company.

7. Release solution
   It is essential that appropriate documentation, training and support are available when the solution is integrated into the normal work processes at the company.

In summary, the technology transfer model focuses on a close collaboration between industry and academia. Through successive validation, it builds trust and commitment to the proposed solution to the industrial challenge identified as suitable for the collaborative effort.

To bring TTM closer to a research methodology, we propose reformulating it as consisting of the following six activities:

1. Identify the industrial challenge
2. Assess practice and formulate a research objective
3. Study state-of-the-art
4. Develop one or more candidate solution(s)
5. Evaluate the solution(s)
   (a) In an academic setting
   (b) Static evaluation
   (c) Pilot evaluation
6. Move the chosen solution into practice and evaluate

Thus, we have divided the second step in TTM into two activities and collapsed the three validation activities into a single evaluation activity. The main reason for the latter is that the three validation steps are viewed as too detailed, and often all three of them are not practiced, as further discussed in Section 6.2. Depending on the technology to be put into practice, the evaluation step may also include training of practitioners. Henceforth, we refer to these six activities as the Technology Transfer Research Methodology (TTRM) to distinguish it from the original TTM. The six activities are easily mapped to the research cycle *describe–solve–practice*.

Given that TTM/TTRM is formulated within software engineering for industry–academia collaboration, we turn directly to the illustration of its use.

6.2. Illustration of TTM/TTRM

TTM has been used as a research methodology in several areas of software engineering and by different researchers. The model emanates from a collaboration between industry and academia in the area of requirements engineering. The model captures how the research resulting in the Requirements Abstraction Model [6] and its evaluation [86] was conducted. Since then, it has been used in software testing for test case selection methods as described by, e.g. Garousi et al. [87] concerning regression testing and de Oliveira Neto et al. [88] in relation to continuous integration. Furthermore, Torkar et al. [89] used TTM for studies on adopting open source in the industry. Moreover, Borg et al. [90] have used it for impact analysis in their industrial collaboration. As a final example in software engineering, we would like to highlight the work by Briand et al. [91] in the area of model–driven engineering. The research by Briand et al. is described in further detail below to illustrate their use of TTM in their research together with industry. It is also worth noting that TTM has been used outside software engineering, which is here exemplified with a study by Pochyly et al. [92] in the area of robotic vision.

Briand et al. [91] presented three research projects in the area of model–driven engineering and their experiences from using an adapted version of the TTM. They adapted TTM by introducing a particular training activity, which we have chosen to integrate into the evaluation activity discussed in Section 6.1. The transfer of novel research solutions was the main driver in the collaboration between industry and academia, and hence training becomes an essential aspect. Furthermore, they only used two validation activities and highlighted that just one might be needed depending on the situation. Thus, TTRM is well-aligned with the use of TTM by Briand et al. References to the activities in TTM as illustrated in Figure 7 are provided below to ease the mapping between the example and the research methodology description.

In their study, Briand et al. [91] used participation in regular project meetings, organizational observations, and meetings and interviews with domain experts to identify and formulate the industrial challenge in their three reported projects (1. Industrial challenge). They continued with studying state–of–the–art about the identified challenge (2. Problem statement and state–of–the–art).

In the next step, they developed candidate solutions, including an information model supporting traceability, a model–based approach for supporting software configuration and a model–based approach for assessment of new technologies (3. Candidate solution). The solutions were based on the identified challenges, the needs of their collaborative partners and available research, and innovative ideas from the participating researchers.

Briand et al. [91] chose to not conduct validation in academia (4. Academic validation). They argue that they did not have a sufficiently good benchmark to conduct a validation in academia. Thus, once candidate solutions were available, they were discussed with the practitioners (5. Static validation), and the solutions were in all three projects evaluated using case studies (6. Dynamic validation). Furthermore, they included training as part of the industrial validation. The research–based innovations have begun to be used, according to Briand et al. [91] (7. Release solution).

Briand et al. have introduced the adapted model as their way of producing research–based innovation together with their industrial partners

7. Comparison

7.1. The three research methodologies

The research methodologies discussed above have been formulated for different main reasons, which we explore below. Thus, the research methodologies are complementary but also competing. They are complementary based on their primary objective when being formulated and competing in the sense that they can all be adapted to produce the main outcome of each other. The primary objective governs to a large extent what each methodology puts forward as its main outcome. This implies that it is not apparent which research methodology to choose.

We recommend that researchers choose wisely and not only use the research methodology they are used to and know very well. The primary objective of each methodology should be an essential aspect. However, elements from other research methodologies may influence the implementation of the chosen research methodology. For example, action research can be used within the methodological frame of design science as a method, see Figure 4. In summary, the main recommendation is to choose based on matching the main goal of your research and your intended way of conducting the research with the primary objective of the research methodology. As a positive side-effect, it may also mean that researchers become more explicit in their primary objective with the research (or study) they conduct.

The primary objective of the three research methodologies may be summarized as follows:

- Design Science Methodologies
  Design science is focused on the design knowledge acquired through the design of artifacts. It emerged as a complement to studying natural phenomena, based on a desire to study artificial or designed constructs. The artifacts primarily focus on tangible outputs and less on social change.

- Action Research
  Action research is focused on social systems and emerged based on a need to put knowledge into action. In action research, the researchers are deeply involved in the change process, i.e. being a member of the team responsible for the change.

- Technology Transfer Research Methodology
  The technology transfer research methodology is an adaptation of the TTM that emerged as a model for collaboration between industry and academia. There is a strong focus on successfully transferring research into practice.

From a software engineering perspective, the different research methodologies may be used in industry–academia collaboration as follows:

- If the primary objective is to derive knowledge through the development of tangible artifacts, then design science methodologies may be a good starting point.

- If the primary objective is to support change, such as changing the ways of working, and the researcher is a member of the team responsible for the change, action research may be a good starting point.

- If the primary objective is to transfer a research result to the industry but not being part of a team responsible for the change. In this case, the technology transfer research methodology may be a good starting point.

It should be noted that the items above are formulated based on having a primary objective in the research. However, it is essential to allow different

	DSM	AR	TTM/TTRM
Origin	Multiple (via IS)	Social science (via IS)	Software engineering
Main outcome	Design knowledge	Change	Knowledge transfer
Roles	Knowledge vs change	Integrated action team	Knowledge vs change
Driver of literature search	Problem and solution	Diagnosing	State-of-the-art study
Learning	Problem and solution theory	Theory as secondary output	Lessons learned

Table 1: Summary of the comparison between the three research methodologies.

research methodologies' strengths to influence the research, although having one research methodology as the main starting point.

Our guidance in selecting a research methodology is based on a systematic comparison between the methodologies used for software engineering research in industry–academia collaboration. We summarize the comparison of the aspects listed in Section 3.3 in Table 1 and elaborate on the similarities and differences below.

The *origin* sets some fundamental principles of the methodologies, which influence the research. The socio-technical characteristics of software engineering imply that methodologies from both a social and technical background are relevant. Furthermore, different branches of software engineering research may focus more on one side or the other.

DSM has multiple roots in the sciences of "the artificial" [40] and has primarily emerged into software engineering via information systems [36, 10], although influences have also come via management science [37]. AR emerged from social science [61] and has also reached software engineering via information systems [71]. The social and information systems origin tend to make the methodologies lean towards the social side, although DSM counteracts that by focusing on artifacts.

TTRM, on the other hand, is designed for software engineering and takes the intangible characteristics of the products and services into account [74].

The *main outcomes* are what the methodology puts forward as the driving force in the research. However, what is the main outcome for one methodology may be secondary outcomes for another. Hence, this is not a clear distinction between what outcomes there may be. It is more a matter of which outcome is considered most important.

DSM focuses on design knowledge, i.e. how to design artifacts and products. The term artifact indicates that DSM is flexible in what type of knowledge may be the outcome, as artifacts may involve product and process-related outcomes [42]. AR focuses on changes in the natural context, similar to TTRM, which focuses on impacting software engineering practice through knowledge transfer.

All methodologies address the collaboration between industry and academia, as they aim to produce (DSM) or transfer knowledge about some practice (TTRM) or make a change of practice (AR). The *roles* taken by researchers and practitioners are, however, slightly different. AR emphasizes the unified research team of both researchers and practitioners, although acknowledging that there are management roles taken by practitioners only. The other two methodologies make more separation in that a joint team, mostly led by researchers, derive the knowledge. At the same time, the practitioners have the ultimate responsibility for the change, based on the derived knowledge. Still, all methodologies aim for industry–academia collaboration and co–production of knowledge, although it is emphasized to somewhat different degrees.

The role of *literature* or underpinning theory as a starting point for the research endeavor differs between the methodologies. DSM, in the Offermann instance (see Figure 4), proposes two literature reviews, one for problem identification and one for solution design. In TTRM, one activity focuses on "study state-of-the-art", identifying potential solutions in the literature. In AR, the literature is more implicitly covered as a part of the diagnosing phase.

The *learning* gained from the research is tightly related to the main outcome in the study contexts. Still, being research methodologies, the learning is also communicated in academic contexts. Venable [12] adds that theory may be output from design science research. According to Staron [5], AR presents theory as a potential outcome, which is in line with general AR [62], although not the main outcome. In TTM/TTRM, which is directed towards supporting engineering, lessons learned are documented, although it is not prescribed to be expressed in terms of theory.

7.2. Implications of comparison

As a socio-technical inter-discipline [93], software engineering is constantly being influenced by multiple research fields. Consequently, increasing awareness and focus on research methodology also leads to influences from adjacent areas, like information systems and other engineering disciplines. In this article, we have analyzed research methodologies from these fields, aiming to derive guidance for what to use in industry–academia collaborative research on software engineering under different circumstances.

Selection of research methodology seems to be a sensitive question, as we have found disputes over research methodology in adjacent fields of research, sometimes conducted in emotional and authoritarian tones [35, 67, 34]. Furthermore, we have observed that many variants of research methodology exist under the same label. When methodologies evolve in parallel in different fields of research, the evolution may take different paths. For example, Offerman et al. [2] analyze five design science variants for information systems and merge them into one. Moreover, regarding action research, one of the multiple variants has been assigned the label 'canonical' [33] as a kind of official mark in an – in this case successful – attempt to homogenize the concepts in a field of research.

Further adding to the confusion about terms and concepts, the labels used for research *methodologies* also appear as a denotation for *paradigms* or *methods*. As presented in Section 4, design science denotes both a paradigm [38, 35] and a methodology [2].

Why are these distinctions important? Aren't they just examples of these "academic battles" without relevance outside a narrow (minded) elite? Unfortunately, smart people have wasted their energy on academic battles, but we argue that the core of these issues is *research conduct* and *communication*. When planning a research endeavor, it is essential that the methodology supports the aims set out for the research. When reviewing a manuscript, it is important to assess it based on its primary claims of contributions.

In our analysis of the three research methodologies and their feasibility for software engineering research, we first observe their similarity. All three match the general research cycle *describe—solve—practice*, as discussed in Section 2 and illustrated in Figure 2. They also, in one way or another, embrace the *knowledge or theory building* activity [12] included in engineering research, in contrast to engineering practice, where the primary focus is to solve a problem rather than understanding why the solution works. Design, change, improvement, transfer, or action are also common elements of the three methodologies. However, what differs between them is what they put forward as primary versus secondary objectives and outcomes, as demonstrated in the comparison in Section 7.1. The roles in the collaboration between researchers and practitioners are also different.

We observe that AR and TTRM primarily focus on change, while DSM primarily aims for more generalized design knowledge. Consequently, AR and TTRM implicitly address one case at a time – although examples of AR studies with two companies exist [79, 80] – while DSM involves multiple cases for generalization. Furthermore, AR stresses the highly integrated action team of researchers and practitioners, while in TTRM, they jointly derive knowledge for practitioners to use in their improvement actions. DSM separates the roles even more in its aim for generalized design knowledge, although it is derived and validated in specific collaboration settings.

In an attempt to combine the aims on equal terms, Wieringa's technical action research [66] includes one engineering cycle and one knowledge cycle, corresponding to AR and DSM, respectively. Similarly, Garousi et al.[87] conducted what they label an AR project, "[u]sing the systematic guidelines for technology transfer provided by Gorschek et al." However, these hybrid approaches risk blurring the primary goal and contribution.

To properly guide the research conduct in industry–academia collaboration and to communicate the outcomes to researchers and practitioners, we, therefore, advice researchers to select research methodology according to their *main goal* of the research. It also helps peer reviewers to assess the contribution properly.

We hope that our guidance, based on an in–depth analysis of the methodologies, will add clarity. Maybe it can encourage some researchers to leave the comfort zone of their favorite methodology, and adopt something new, that better fits the main goal of their research?

8. Conclusion

The selection of appropriate research methodologies in solution–oriented software engineering research in close collaboration between industry and academia is essential. The selection of a research methodology should help ensuring successful collaborative research between industry and academia. It is vital both for the research conduct and the communication of its results. As software engineering is inter–disciplinary, methodological influences come from different disciplines, and may be used and referred to in different ways.

We selected three methodologies to bring clarity about research methodologies and provide guidance for their selection in industry–academia collaboration. They are used in software engineering and cover various characteristics, namely DSM, AR and TTRM, and they were analyzed theoretically, using thematic synthesis.

The research methodologies can all be characterized (RQ1) as cyclic research processes, aligning to the general research cycle of *describe—solve—practice*.

The second research question (RQ2) encompasses both similarities and differences. Concerning similarities, the methodologies are similar in addressing generalized learning or theorizing in the collaboration between researchers and practitioners, although in different forms, using different terminology. All methodologies have been used in software engineering research. It is also worth noting that the same methods for data collection (for example, surveys and observations) and analysis (statistical modeling and thematic analysis) may be used in all of the methodologies.

When it comes to differences, the three methodologies differ in their primary objective: DSM on acquiring design knowledge through the design of artifacts, AR on change in socio–technical systems, and TTRM on the transfer of research to industry. The primary objective of one methodology may be a secondary objective in another. Thus, the differences between them are more in their focus than in which activities they include.

In our analysis and comparison of their feasibility for industry–academia collaboration in software engineering research, the selection depends on the primary objective and scope of the research (RQ3). We, therefore, advice researchers to consider the objectives of their software engineering research endeavor and select an appropriate methodological frame accordingly. Furthermore, we recommend studying different sources of information concerning, in particular, the chosen research methodology to better understand the methodology before using it when conducting industry–academia collaborative research.

Acknowledgments

The work is partially supported by a research grant for the ELLIIT strategic research area funded by the Swedish Government. We thank the anonymous reviewers for their valuable feedback. The comments provided helped make the scope clearer and improve the paper.

Appendix A. Supplementary data

A link to supplementary material, related to the Design Research Methodology, can be found in this section in the online version of the article at: **Link**

to be provided by Elsevier

References

[1] T. S. Kuhn, The Structure of Scientific Revolutions, The University of Chicago Press, 1962.

[2] P. Offermann, O. Levina, M. Schönherr, U. Bub, Outline of a design science research process, in: Proceedings International Conference on Design Science Research in Information Systems and Technology, ACM, 2009, pp. 1–11.

[3] R. J. Wieringa, Design Science Methodology for Information Systems and Software Engineering, Springer Berlin Heidelberg, 2014.

[4] R. L. Baskerville, T. Wood-Harper, Diversity in information systems action research methods, European Journal of Information Systems 7 (1998) 90–107.

[5] M. Staron, Action Research in Software Engineering - Theory and Applications, Springer, 2020.

[6] T. Gorschek, C. Wohlin, Requirements abstraction model, Requirements Engineering Journal 11 (2006) 79–101.

[7] T. Gorschek, K. Wnuk, Third generation industrial co-production in software engineering, in: M. Felderer, G. H. Travassos (Eds.), Contemporary Empirical Methods in Software Engineering, Springer, 2020, pp. 503–525.

[8] M. Shaw, Prospects for an engineering discipline of software, IEEE Software 7 (1990) 15–24.

[9] M. Shaw, Research toward an engineering discipline for software, in: G. Roman, K. J. Sullivan (Eds.), Proceedings Workshop on the Future of Software Engineering Research, ACM, 2010, pp. 337–342.

[10] R. Wieringa, Design science as nested problem solving, in: Proceedings International Conference on Design Science Research in Information Systems and Technology, ACM, 2009, pp. 8:1–8:12.

[11] N. M. Agnew, S. W. Pyke, The Science Game – An Introduction to Research in the Social Sciences, Oxford University Press, 7th edition, 2007.

[12] J. R. Venable, The role of theory and theorising in design science research, in: Proceedings Design Science Research in Information Systems and Technology, Curtin Research Publications, 2006, pp. 1–18.

[13] V. Garousi, K. Petersen, B. Ozkan, Challenges and best practices in industry-academia collaborations in software engineering: A systematic literature review, Information and Software Technology 79 (2016) 106–127.

[14] J. Brings, M. Daun, S. Brinckmann, K. Keller, T. Weyer, Approaches, success factors, and barriers for technology transfer in software engineering – results of a systematic literature review, Journal of Software: Evolution and Process 30 (2018) e1981.

[15] C. Wohlin, A. Aurum, L. Angelis, L. Phillips, Y. Dittrich, T. Gorschek, H. Grahn, K. Henningsson, S. Kågström, G. Low, P. Rovegård, P. Tomaszewski, C. V. Toorn, J. Winter, The success factors powering industry-academia collaboration, IEEE Software 29 (2012) 67–73.

[16] T. Mikkonen, C. Lassenius, T. Männistö, M. Oivo, J. Järvinen, Continuous and collaborative technology transfer: Software engineering research with real-time industry impact, Information and Software Technology 95 (2018) 34–45.

[17] K. Beckman, N. Coulter, S. Khajenoori, N. R. Mead, Collaborations: closing the industry-academia gap, IEEE Software 14 (1997) 49–57.

[18] A. Sandberg, L. Pareto, T. Arts, Agile collaborative research: Action principles for industry-academia collaboration, IEEE Software 28 (2011) 74–83.

[19] K. Stol, B. Fitzgerald, The ABC of software engineering research, ACM Transactions on Software Engineering and Methodology 27 (2018) 11:1–11:51.

[20] M.-A. Storey, N. A. Ernst, C. Williams, E. Kalliamvakou, The who, what, how of software engineering research: a socio-technical framework, Empirical Software Engineering 25 (2020) 4097–4129.

[21] M. Felderer, G. H. Travassos (Eds.), Contemporary Empirical Methods in Software Engineering, Springer, 2020.

[22] F. Shull, J. Singer, D. I. K. Sjøberg (Eds.), Guide to Advanced Empirical Software Engineering, Springer, 2008.

[23] P. Runeson, M. Höst, A. Rainer, B. Regnell, Case Study Research in Software Engineering – Guidelines and Examples, Wiley, 2012.

[24] S. Easterbrook, J. Singer, M.-A. Storey, D. Damian, Selecting empirical methods for software engineering research, in: F. Shull, J. Singer, D. I. K. Sjøberg (Eds.), Guide to Advanced Empirical Software Engineering, Springer London, London, 2008, pp. 285–311.

[25] C. Wohlin, A. Aurum, Towards a decision-making structure for selecting a research design in empirical software engineering, Empirical Software Engineering 20 (2015) 1427–1455.

[26] L. Briand, D. Bianculli, S. Nejati, F. Pastore, M. Sabetzadeh, The case for context-driven software engineering research: Generalizability is overrated, IEEE Software 34 (2017) 72–75.

[27] V. Basili, L. Briand, D. Bianculli, S. Nejati, F. Pastore, M. Sabetzadeh, Software engineering research and industry: A symbiotic relationship to foster impact, IEEE Software 35 (2018) 44–49.

[28] P. Runeson, S. Minör, J. Svenér, Get the cogs in synch – time horizon aspects of industry–academia collaboration, in: International Workshop on Long-term Industrial Collaboration on Software Engineering, ACM, 2014, pp. 25–28.

[29] C. Wohlin, E. Mendes, K. R. Felizardo, M. Kalinowski, Challenges and recommendations to publishing and using credible evidence in software engineering, Information and Software Technology 127 (2020) 106366.

[30] K.-J. Stol, P. Ralph, B. Fitzgerald, Grounded theory in software engineering research: A critical review and guidelines, in: Proceedings International Conference on Software Engineering, ACM, 2016, pp. 120–131.

[31] H. Sharp, Y. Dittrich, C. R. B. de Souza, The role of ethnographic studies in empirical software engineering, IEEE Transactions on Software Engineering 42 (2016) 786–804.

[32] D. S. Cruzes, T. Dybå, P. Runeson, M. Höst, Case studies synthesis: A thematic, cross-case, and narrative synthesis worked example, Empirical Software Engineering 20 (2015) 1634–1665.

[33] R. M. Davison, M. G. Martinsons, N. Kock, Principles of canonical action research, Information Systems Journal 14 (2004) 65–86.

[34] R. L. Baskerville, What design science is not, European Journal of Information Systems 17 (2008) 441–443.

[35] J. Iivari, J. Venable, Action research and design science research - seemingly similar but decisively dissimilar, in: S. Newell, E. A. Whitley, N. Pouloudi, J. Wareham, L. Mathiassen (Eds.), Proceedings European Conference on Information Systems, Association for Information Systems, 2009, pp. 1642–1653.

[36] A. R. Hevner, S. T. March, J. Park, S. Ram, Design science in information systems research, MIS Quarterly 28 (2004) 75–105.

[37] J. E. van Aken, 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 (2004) 219–246.

[38] P. Runeson, E. Engström, M.-A. Storey, The design science paradigm as a frame for empirical software engineering, in: M. Felderer, G. H. Travassos (Eds.), Contemporary Empirical Methods in Software Engineering, Springer, 2020, pp. 129–149.

[39] E. Engström, M.-A. Storey, P. Runeson, M. Höst, M. T. Baldassarre, How software engineering research aligns with design science: A review, Empirical Software Engineering 25 (2020) 2630–2660.

[40] H. A. Simon, The Sciences of the Artificial, MIT Press, 1969.

[41] K. Peffers, T. Tuunanen, M. A. Rothenberger, S. Chatterjee, A design science research methodology for information systems research, Journal of Management Information Systems 24 (2007) 45–77.

[42] P. Offermann, S. Blom, M. Schönherr, U. Bub, Artifact types in information systems design science – a literature review, in: R. Winter, J. L. Zhao, S. Aier (Eds.), Global Perspectives on Design Science Research, Springer Berlin Heidelberg, 2010, pp. 77–92.

[43] V. Garousi, M. Felderer, M. V. Mäntylä, Guidelines for including grey literature and conducting multivocal literature reviews in software engineering, Information and Software Technology 106 (2019) 101–121.

[44] B. A. Kitchenham, D. Budgen, P. Brereton, Evidence-Based Software Engineering and Systematic Reviews, Chapman and Hall/CRC, 2015.

[45] J. Kontio, J. Bragge, L. Lehtola, The focus group method as an empirical tool in software engineering, in: F. Shull, J. Singer, D. I. K. Sjøberg (Eds.), Guide to Advanced Empirical Software Engineering, Springer London, London, 2008, pp. 93–116.

[46] P. Runeson, M. Höst, Guidelines for conducting and reporting case study research in software engineering, Empirical Software Engineering 14 (2009) 131–164.

[47] N. H. Thuan, A. Drechsler, P. Antunes, Construction of design science research questions, Communications of the Association for Information Systems 44 (2019) 332–363.

[48] P. Johannesson, E. Perjons, An Introduction to Design Science, Springer, 2014.

[49] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in Software Engineering, Springer, 2012.

[50] C. Wohlin, Guidelines for snowballing in systematic literature studies and a replication in software engineering, in: Proceedings International Conference on Evaluation and Assessment in Software Engineering, ACM, 2014, p. 38.

[51] C. Wohlin, A. Rainer, Guidelines for the search strategy to update systematic literature reviews in software engineering, Information and Software Technology 134 (2021) 106555.

[52] K. Stol, B. Fitzgerald, Guidelines for conducting software engineering research, in: M. Felderer, G. H. Travassos (Eds.), Contemporary Empirical Methods in Software Engineering, Springer, Cham, 2020, pp. 27–63.

[53] R. J. Wieringa, Design science methodology: principles and practice, in: J. Kramer, J. Bishop, P. T. Devanbu, S. Uchitel (Eds.), Proceedings International Conference on Software Engineering, ACM, 2010, pp. 493–494.

[54] D. Méndez Fernández, B. Penzenstadler, Artefact-based requirements engineering: the AMDiRE approach, Requirements Engineering 20 (2015) 405–434.

[55] A. Morali, R. J. Wieringa, Risk-based confidentiality requirements specification for outsourced IT systems, in: Proceedings International Requirements Engineering Conference, IEEE Computer Society, 2010, pp. 199–208.

[56] T. O. A. Lehtinen, M. Mäntylä, J. Vanhanen, Development and evaluation of a lightweight root cause analysis method (ARCA method) - field studies at four software companies, Information and Software Technology 53 (2011) 1045–1061.

[57] E. Bjarnason, H. Sharp, B. Regnell, Improving requirements-test alignment by prescribing practices that mitigate communication gaps, Empirical Software Engineering 24 (2019) 2364–2409.

[58] C. R. Gumiran, J. M. Gumiran, Applying design science research in the development of human resource record management system with predictive analysis through pointing system, in: Proceedings International Conference on Software and Computer Applications, ACM, 2019, pp. 243–247.

[59] P. Pickerill, H. J. Jungen, M. Ochodek, M. Mackowiak, M. Staron, PHANTOM: curating GitHub for engineered software projects using time-series clustering, Empirical Software Engineering 25 (2020) 2897–2929.

[60] E. Bjarnason, K. Smolander, E. Engström, P. Runeson, A theory of distances in software development, Information and Software Technology 70 (2016) 204–219.

[61] K. Lewin, Action research and minority problems, Journal of Social Issues 2 (1946) 34–46.

[62] G. I. Susman, R. D. Evered, An assessment of the scientific merits of action research, Administrative Science Quarterly 23 (1978) 582–603.

[63] I. Chein, S. W. Cook, J. Harding, The field of action research, American Psychologist 3 (1948) 43–50.

[64] T. Wood-Harper, Research methods in information systems: Using action research, in: E. Mumford, R. A. Hirschheim, G. Fitzgerald, T. Wood-Harper (Eds.), Research Methods in Information Systems, Elsevier Science Publishers B.V., North-Holland, 1985, pp. 161–180.

[65] P. S. M. dos Santos, G. H. Travassos, Action research can swing the balance in experimental software engineering, in: M. V. Zelkowitz (Ed.), Advances in Computers, volume 83, Elsevier, 2011, pp. 205–276.

[66] R. J. Wieringa, A. Moralı, Technical action research as a validation method in information systems design science, in: Proceedings Design Science Research in Information Systems. Advances in Theory and Practice, Springer, Berlin, Heidelberg, 2012, pp. 220–238.

[67] P. Järvinen, Action research is similar to design science, Quality & Quantity 41 (2007) 37—54.

[68] S. Kemmis, R. McTaggart, The Action Research Planner, Action research and the critical analysis of pedagogy, Deakin University, 1988.

[69] C. Andersson, P. Runeson, A spiral process model for case studies on software quality monitoring - Method and metrics, Software Process Improvement and Practice 12 (2007) 125–140.

[70] B. W. Boehm, A spiral model of software development and enhancement, IEEE Computer 21 (1988) 61–72.

[71] R. L. Baskerville, Investigating information systems with action research, Communications of the Association for Information Systems 2:19 (1999) 2–31.

[72] C. Wohlin, D. Smite, N. B. Moe, A general theory of software engineering: Balancing human, social and organizational capitals, Journal of Systems and Software 109 (2015) 229–242.

[73] K. Petersen, C. Gencel, N. Asghari, D. Baca, S. Betz, Action research as a model for industry-academia collaboration in the software engineering context, in: Proceedings International Workshop on Long-Term Industrial Collaboration on Software Engineering, ACM, 2014, pp. 55–62.

[74] V. Garousi, M. Borg, M. Oivo, Practical relevance of software engineering research: synthesizing the community's voice, Empirical Software Engineering 25 (2020) 1687–1754.

[75] J. C. Carver, R. Prikladnicki, Industry–academia collaboration in software engineering, IEEE Software 35 (2018) 120–124.

[76] C. B. Seaman, Qualitative methods in empirical studies of software engineering, IEEE Transactions on Software Engineering 25 (1999) 557–572.

[77] J. E. Hannay, D. I. K. Sjøberg, T. Dybå, A systematic review of theory use in software engineering experiments, IEEE Transactions on Software Engineering 33 (2007) 87–107.

[78] K. Stol, B. Fitzgerald, Theory-oriented software engineering, Science of Computer Programming 101 (2015) 79–98.

[79] M. Staron, M. Ochodek, W. Meding, O. Söder, Using machine learning to identify code fragments for manual review, in: Proceedings Euromicro Conference on Software Engineering and Advanced Applications, IEEE, 2020, pp. 513–516.

[80] M. Ochodek, R. Hebig, W. Meding, G. Frost, M. Staron, Recognizing lines of code violating company-specific coding guidelines using machine learning, Empirical Software Engineering 25 (2020) 220–265.

[81] M. Choras, T. Springer, R. Kozik, L. López, S. Martínez-Fernández, P. Ram, P. Rodríguez, X. Franch, Measuring and improving agile processes in a small-size software development company, IEEE Access 8 (2020) 78452–78466.

[82] M. Razavian, P. Lago, A lean and mean strategy: a data migration industrial study, Journal of Software: Evolution and Process 26 (2014) 141–171.

[83] A. Ananjeva, J. S. Persson, A. Bruun, Integrating UX work with agile development through user stories: An action research study in a small software company, Journal of Systems and Software 170 (2020) 110785.

[84] T. Gorschek, P. Garre, S. Larsson, C. Wohlin, A model for technology transfer in practice, IEEE Software 23 (2006) 88–95.

[85] N. B. Ali, K. Petersen, C. Wohlin, A systematic literature review on the industrial use of software process simulation, Journal of Systems and Software 97 (2014) 65–85.

[86] T. Gorschek, P. Garre, S. Larsson, C. Wohlin, Industry evaluation of the requirements abstraction model, Requirements Engineering Journal 12 (2007) 163–190.

[87] V. Garousi, R. Özkan, A. Betin-Can, Multi-objective regression test selection in practice: An empirical study in the defense software industry, Information and Software Technology 103 (2018) 40–54.

[88] F. G. de Oliveira Neto, A. Ahmad, O. Leifler, K. Sandahl, E. Enoiu, Improving continuous integration with similarity-based test case selection, in: Proceedings International Workshop on Automation of Software Test, ACM, 2018, pp. 39–45.

[89] R. Torkar, P. Minoves, J. Garrigós, Adopting free/libre/open source software practices, techniques and methods for industrial use, Journal of the Association for Information Systems 12 (2011) 88–122.

[90] M. Borg, K. Wnuk, B. Regnell, P. Runeson, Supporting change impact analysis using a recommendation system: An industrial case study in a safety-critical context, IEEE Transactions on Software Engineering 43 (2017) 675–700.

[91] L. Briand, D. Falessi, S. Nejati, M. Sabetzadeh, T. Yue, Research-based innovation: A tale of three projects in model-driven engineering,

in: R. B. France, J. Kazmeier, R. Breu, C. Atkinson (Eds.), Proceedings Model Driven Engineering Languages and Systems, Springer Berlin Heidelberg, 2012, pp. 793–809.

[92] A. Pochyly, T. Kubela, M. Kozak, P. Cihak, Robotic vision for bin-picking applications of various objects, in: Proceedings International Symposium on Robotics) and German Conference on Robotics), pp. 1–5.

[93] D. Méndez Fernández, J.-H. Passoth, Empirical software engineering: From discipline to interdiscipline, Journal of Systems and Software 148 (2019) 170–179.

Appendix A: DRM - Design Research Methodology

This appendix provides an overview and discussion of the Design Research Methodology (DRM). Furthermore, DRM is compared with the three research methodologies presented in the main article, i.e. Design Science Methodology (DSM), Action Research (AR) and the Technology Transfer Research Methodology (TTRM).

DRM is provided as an add-on here based on that we view software engineering as an engineering discipline. Thus, we looked at research methodologies in other engineering disciplines. The research methodology that stood out when looking at the literature and discussing research methodologies with engineering colleagues was the Design Research Methodology (DRM). Hence, it was included among the methodologies analyzed and compared. However, due to space constraints and the limited use of DRM in software engineering, we provide it as complementary material.

A1. Overview of Design Research Methodology

Blessing and Chakrabarti present the Design Research Methodology (DRM) for research on various kinds of design of products [1]. The design research methodology is design-oriented but is grounded in engineering and particular product design, development and innovation. Blessing and Chakrabarti point to three issues in design research: "i) a lack of overview of existing research, ii) a lack of use of the research in practice, and iii) a lack of scientific rigor." According to Blessing and Chakrabarti [1], the main objective of DRM is to address the lack of scientific rigor. They believe that addressing the issue concerning scientific rigor will also contribute to addressing the two other issues. The book by Blessing and Chakrabarti [1] is classified as a book in mechanical engineering by the publisher. Design of products is defined as "those activities that actually generate and develop a product from a need, product idea or technology to the full documentation needed to realize the product and to fulfill the perceived needs of the user and other stakeholders." [1]. Three questions define the overall aims:

- What do we mean by a successful product?
- How is a successful (or unsuccessful) product created?
- How do we improve the chances of being successful?

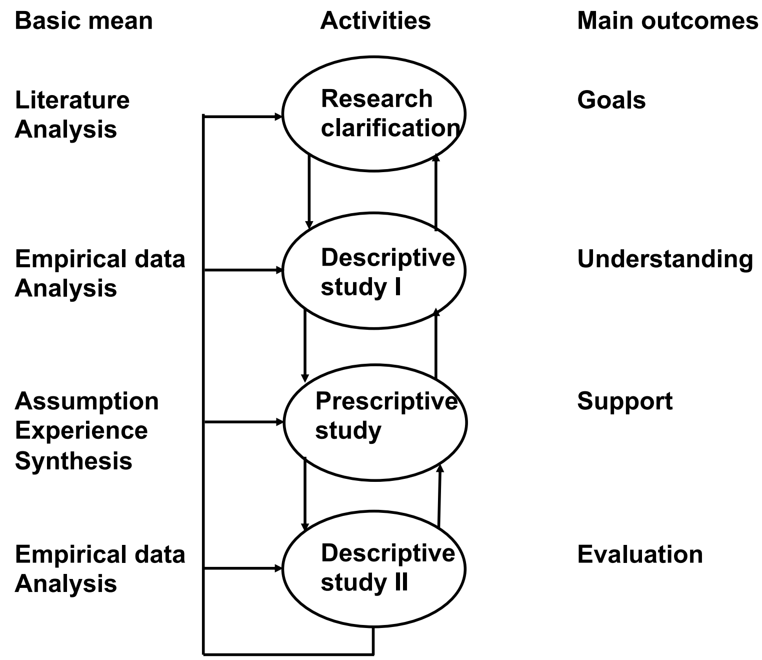

Figure 8: DRM adapted from the book by Blessing and Chakrabarti [1].

The design research methodology is iterative over four stages 1) Research Clarification for the setting goals, 2) Descriptive Study I to increase the understanding of the research challenge, 3) Prescriptive Study for supporting the product design, and 4) Descriptive Study II for evaluation of the outcome. The stages with the basic means to obtain the outcomes in each stage and the primary outcomes are illustrated in Figure 8.

Furthermore, three types of studies are defined: *review-based, comprehensive* and *initial*. The review-based study focuses on conducting a literature review. The comprehensive study includes both a literature review and a study where the researchers produce results, for example, an empirical study. From a usage point of view, the initial study closes a research project and is focused on illustrating the consequences of the findings and preparing the findings for use by others. Thus, the word “initial” refers to the initial application of the solution and not the initial research. The research clarification stage is typically conducted as a review-based study. The other stages could be of different types. Overall, Blessing and Chakrabarti [1] describe seven types of design of research projects where the four stages are conducted in different ways using the three types of studies. The seven types of design are listed in Table 2.

Table 2: The seven design types in DRM.

Research clarification	**Descriptive study I**	**Prescriptive study**	**Descriptive study II**
1. Review-based	Comprehensive		
2. Review-based	Comprehensive	Initial	
3. Review-based	Review-based	Comprehensive	Initial
4. Review-based	Review-based	Review-based Initial/ Comprehensive	Comprehensive
5. Review-based	Comprehensive	Comprehensive	Initial
6. Review-based	Review-based	Comprehensive	Comprehensive
7. Review-based	Comprehensive	Comprehensive	Comprehensive

The rows in Table 2 should be interpreted as follows:

- Design types 1-3 and 5 are conducted from left to right without iterations.

- Design type 4 includes an iteration from the Comprehensive study conducted in the Descriptive study II activity to the Descriptive study when either an Initial or Comprehensive study is undertaken. It is then followed by revisiting the Descriptive study II stage.

- Design types 6 and 7 includes more iterations. From the Comprehensive study in the Descriptive study II stage, we may go back to either Descriptive study I or the Prescriptive study stage and then continue with the following stages and iterate until no further improvements of the solution are envisioned.

DRM includes many alternatives with its four stages and three study types, particularly when including the possibility of iterations. Motivations for the seven design types and further information about the use of DRM are provided in Blessing and Chakrabarti [1].

Blessing and Chakrabarti [1] describe how the first four design types of research projects in Figure 8 are particularly suitable for PhD projects. Based on their experience, types 2 and 3 are the most common. Design types 5 and 6 include two comprehensive studies, and based on their experience, it is most often a too large undertaking for a PhD project. Even if the aim is types 5 and 6, they mostly end up being of type 2 or type 3. Type 7 includes three comprehensive studies, which may be more suitable for a research team than for an individual researcher or when the scope is very specific.

Furthermore, Blessing and Chakrabarti [1] also introduce two types of models: the *reference model* and the *impact model*. The reference model is intended to represent the current state, i.e. before conducting the research, while the impact model illustrates the desired state after the research. Both models are revised as the understanding of the research context increases and the solution is evolving.

In summary, the design research methodology involves a combination of stages, different types of studies, including combining them in different ways, and models to capture the state of the situation both before and after the research.

A2. DRM for industry–academia collaboration in software engineering

The Design Research Methodology is a methodology for a research project rather than a methodology for a research study. Blessing and Chakrabarti [1] discuss, for example, the use of the research methodology for a PhD student and hence outlining the research work during the PhD studies. Blessing and Chakrabarti [1] highlight the lack of use of research in practice (see Section A.1). Thus, their standpoint is that DRM is well suited to be used in industry–academia collaborative research. DRM is easily mapped to the general research cycle for engineering in Figure 2. Hence, it is straightforward to map it to the describe–solve–practice activities used in the main article. DRM starts with setting the goals and creating an understanding of the problem at hand, as shown in Figure 8, which is well-aligned with describing the current situation. It is followed by a prescriptive study (support), which concerns the development of a solution, which relates to solve. Finally, the solution is evaluated, which maps directly to the general problem-solving cycle.

Blessing and Chakrabarti [1] compare DRM with a DRM predecessor by Duffy and Andreasen [2] and to the soft systems methodology introduced by Checkland [3]. The latter is described as action research by Blessing and Chakrabarti. The main difference, according to them, is the focus of action research in being on-site evaluation, which results in more local solutions vs providing more general solutions using DRM.

From a software engineering perspective, DRM may be applied as described by Blessing and Chakrabarti [1]. However, it does require that solutions, to a large extent, can be evaluated in the research environment. The main reason being that the objective is to provide more general solutions, as highlighted by Blessing and Chakrabarti [1] when comparing to the soft systems methodology [3].

In many cases, this may be challenging in software engineering, where improvements often relate to the software development process. Given that DRM primarily was developed for product design, there is a need to consider how it is best applied in software engineering, particularly in research conducted in close collaboration between industry and academia. In a software engineering context, new research solutions will inevitably interact with other parts of the processes, methods and tools used in practice. Consequently, the effect of new research solutions most often need to be evaluated in a broader and more realistic context than what can be done in the research environment as such.

The application of DRM in software engineering is limited so far. A search on Google Scholar resulted in identifying only one paper. It was a paper by Eklund and Bosch on open software ecosystems for embedded systems [4]. Each of the four stages in DRM is addressed in separate sections in the paper by Eklund and Bosch [1].

In addition to being used on its own in software engineering, DRM may potentially be combined with other research methodologies for research studies. This is further elaborated in Section 7 in the main article, where the different research methodologies are compared, and possibilities for combing research methodologies are discussed.

A3. Comparison of Design Research Methodology

This section is best read together with Section 7 on comparing the research methodologies in the main article.

The primary objective of DRM is as follows:

- Design Research Methodology
  The design research methodology emerged from a need to make engineering research more scientific. It is product-oriented, which may be interpreted as having the design of artifacts as its primary objective. The methodology is focused on general solutions developed by conducting several studies with different objectives.

DRM is suitable if the primary objective is a general solution to a research problem, although being identified in a collaborative effort between industry

	DSM	AR	TTM/TTRM	DRM
Origin	Multiple (via IS)	Social science (via IS)	Software engineering	Mechanical engineering
Main outcome	Design knowledge	Change	Knowledge transfer	Engineering knowledge
Roles	Knowledge vs. change	Integrated action team	Knowledge vs. change	Knowledge vs. change
Driver of literature search	Problem and solution	Diagnosing	State-of-the-art study	Separate or part of problem/solution depending on design type
Learning	Problem and solution theory	Theory as secondary output	Lessons learned	Lessons learned

Table 3: Summary of the comparison between the four research methodologies.

and academia. DRM is particularly suitable if the research contains a series of studies leading to the general research outcome. In this situation, the design research methodology may be a good starting point. The main outcome may be product design knowledge or extended to other types of design.

Table 3 is copied from the main article to simplify the understanding of DRM in relation to Design Science Methodologies, Action Research and the Technology Transfer Research Methodology.

Concerning the table, the following specific comments concerning DRM are provided.

- Origin – DRM is an engineering methodology for research on design of products, primarily in mechanical engineering [1]. While the principles may apply to any kind of engineering, the mechanical context colors the methodology.

- Main outcome – DSM and DRM both produce design knowledge, i.e. how to design artifacts and products, respectively.

- Roles – DRM address the collaboration between research and practice, as it aims to produce. As AR and TTM/TTRM, DRM makes a separation in that a joint team, mostly led by researchers, derive the knowledge, while the practitioners have the ultimate responsibility for the change, based on the derived knowledge.

- Driver of literature search – Similarly to DSM, DRM, which comes in different configurations (i.e. design types which are described in Table 2) opens for "review-based" studies, either as a separate study to clarify the research needs or as a part of a comprehensive study.

- Learning – DRM explicitly proposes three different publication units: 1) problem identification for practitioners, 2) lab experiments, and 3) case studies for academics. As for TTM/TTRM, DRM is directed towards supporting engineering, lessons learned are documented, although it is not prescribed to be expressed in terms of theory.

Finally, it is worth noting that DRM is a framework for a research project, and thus activities are defined at a level where a research study is an activity.

References

1. L. T. M. Blessing, A. Chakrabarti, DRM, a Design Research Methodology, Springer Publishing Company, Incorporated, 1st edition, 2009.
2. A. H. B. Duffy, M. M. Andreasen, Enhancing the evolution of design science, in: V. Hubka (Ed.), Proceedings of the 10th International Conference on Engineering Design (ICED), Heurista, Zurich, Switzerland,1995, pp. 29–35.
3. P. Checkland, Systems Thinking, Systems Practice, John Wiley and Sons Ltd., Chichester, UK, 1981.
4. U. Eklund, J. Bosch, Architecture for embedded open software ecosystems, Journal of Systems and Software 92 (2014) 128–142.