Received: 14 May 2020 | Revised: 13 January 2021 | Accepted: 18 January 2021

DOI: 10.1002/smr.2345

RESEARCH ARTICLE - METHODOLOGY

Journal of Software: Evolution and Process WILEY

Towards evidence-based decision-making for identification and usage of assets in composite software: A research roadmap

Claes Wohlin[1] | Efi Papatheocharous[2] | Jan Carlson[3] | Kai Petersen[1,4] | Emil Alégroth[1] | Jakob Axelsson[2,3] | Deepika Badampudi[1] | Markus Borg[2] | Antonio Cicchetti[3] | Federicio Ciccozzi[3] | Thomas Olsson[2] | Séverine Sentilles[3] | Mikael Svahnberg[1] | Krzysztof Wnuk[1] | Tony Gorschek[1]

[1]Department of Software Engineering, Blekinge Institute of Technology, Karlskrona, Sweden

[2]Software and Systems Engineering Laboratory, RISE Research Institutes of Sweden AB, Kista, Sweden

[3]Division of Computer Science and Software Engineering, Mälardalen University, Västerås, Sweden

[4]Fachbereich Wirtschaft, University of Applied Sciences Flensburg, Flensburg, Germany

Correspondence
Kai Petersen, Department of Software Engineering, Blekinge Institute of Technology, Karlskrona, Sweden.
Email: kai.petersen@bth.se

Funding information
Knowledge Foundation, Grant/Award Number: 20140218

Abstract

Software engineering is decision intensive. Evidence-based software engineering is suggested for decision-making concerning the use of methods and technologies when developing software. Software development often includes the reuse of software assets, for example, open-source components. Which components to use have implications on the quality of the software (e.g., maintainability). Thus, research is needed to support decision-making for composite software. This paper presents a roadmap for research required to support evidence-based decision-making for choosing and integrating assets in composite software systems. The roadmap is developed as an output from a 5-year project in the area, including researchers from three different organizations. The roadmap is developed in an iterative process and is based on (1) systematic literature reviews of the area; (2) investigations of the state of practice, including a case survey and a survey; and (3) development and evaluation of solutions for asset identification and selection. The research activities resulted in identifying 11 areas in need of research. The areas are grouped into two categories: areas enabling evidence-based decision-making and those related to supporting the decision-making. The roadmap outlines research needs in these 11 areas. The research challenges and research directions presented in this roadmap are key areas for further research to support evidence-based decision-making for composite software.

KEYWORDS

asset origins, component-based software engineering (CBSE), decision-making, evidence-based software engineering, software architecture

1 | INTRODUCTION

The amount of functionality delivered by software has had a tremendous growth in recent decades in every domain, from automotive and telecom to manufacturing, medical equipment, and a range of other areas. The system architectures comprising today's products are no longer monolithic. They are complex, made up of heterogeneous parts, and composed of different architectural patterns and components that can be more or less loosely coupled. Thus, one of the key enablers for successful development is external asset reuse, such as libraries, frameworks, components, and services, or reuse of software assets developed internally.[1] *Software assets* is used as a collective term for software parts that may be reusable. Furthermore, the life cycle of products can differ substantially, the development process can involve many different companies (i.e., suppliers, integrators, and manufacturers), and various components of a system can be developed under different circumstances.

Although composing different software assets in software development is paramount to the efficient development of modern software systems, it also introduces a new set of important decisions and trade-offs related to the origin of the different assets comprising the software being developed. Should we develop a new authentication service, adjust the solution developed for a previous product, or convert an existing open-source solution for the specific needs of the new system? Which of the available databases are most suitable for the system we are developing, in terms of functionality, performance and cost, but also for example long-term support and maintenance?

Even though some of these decisions can have very severe and long-term implications on the qualities of a software product, industrial decision-making in the area of software development typically does not follow an explicit process, decisions are often subjective and based on gut feeling[2] and decisions and their rationales are rarely documented.[3] Decision-making related to software design and architecture is often viewed as a mostly rational process, where a number of alternatives are identified and evaluated with respect to given criteria to determine the most suitable alternative (see, e.g., Ruhe[4]). Recent research, however, emphasizes the bounded rationality of decision-making and stresses human factors such as Groupthink and Anchoring bias.[5] We recognize the complexity of decision-making and agree that fully rational decision-making is neither possible nor desired. However, for software development to evolve into more systematic practices, allowing for high (and predictable) quality, within the expected timeframe and at the expected cost, we believe that important decisions, in particular decisions of strategic, operational or tactical importance,[6,7] need to be more firmly based on evidence than is currently the state of practice. We formulate the following vision:

VISION—**Decision-making in software engineering will be evidence-based**.

Evidence-based software engineering as a concept was introduced in 2004, inspired by the medical sciences, by Kitchenham et al.[8] Dybå et al.[9] suggest that practitioners should consider evidence-based software engineering and their decision-making to be informed by evidence. The concept has created interest, for example, in the form of systematic literature reviews[10] aiming at summarizing the knowledge in specific areas. However, it has not (yet) to any larger extent made it into software practice, based on our experiences conducting research in close collaboration with companies and also noted by Zannier et al.[2] Based on experience from healthcare, Nowak and Schünemann reflect upon how evidence-based decisions also could be beneficial in software engineering.[11] When practicing an evidence-based approach, they stress the need for combining evidence with professional judgment, that is, supporting the professional decision-making with evidence.

Here, we present a roadmap with a particular focus on decisions *related to the origin of software assets when developing composite software systems*. This concerns strategic, tactical and operational decisions. In particular, strategic decisions deal with deciding among options such as open-source software (OSS) versus commercial off-the-shelf components (COTS components) versus in-house development. Tactical decisions are concerned with choosing a concrete component (e.g., which COTS components to use). Operational decisions focus on how to integrate the asset in the composite software system and how to verify the system after the integration of assets from different sources.

We identify, and motivate, a number of core challenges that need to be addressed to advance the state of the art as well as state of practice in the direction of the vision, and suggest research directions of particular relevance in the respective areas.

The remainder of the paper is structured as follows. Section 2 describes related work and motivates the need for the roadmap. In Section 3, the method for identification and definition of the areas needing further research concerning evidence-based decision-making for composite software is presented. Section 4puts the areas into context and lists the areas identified. Section 5 describes the areas concerning challenges concerning decision-making, and Section 6 introduces the areas in need of research to support the decision-making as such. Section 7 presents a discussion putting the different areas into a broader context. The paper is concluded in Section 8.

2 | RELATED WORK

2.1 | Research roadmaps

Research roadmaps are intended to point to research needed in an area to make further progress. In the proceedings of the the track Future of Software Engineering at the International Software Engineering Conference (ICSE) in the year 2000,[12] 26 roadmaps were published covering software engineering in general and numerous areas within software engineering. Since then, roadmaps are published regularly either updating previous roadmaps[13] or adding new roadmaps[14] due to the evolution of technology and the software engineering discipline as such.

2.2 | Composite software

Research directions for software composition were formulated already in 1995 by Nierstrasz and Meijler,[15] and hence the area of software composition is not new. However, the area has evolved substantially since then, including for example, the evolution of open-source software development and technology advancement making use of software assets from anywhere possible.

The interest in better comprehending how composite software architecture is designed arose with critical analysis and identification of patterns and paradigms such as Service Oriented Architecture (SOA), architectural styles, and reusable, parameterized reference architectures many years ago (cf. Shaw and Garlan[16] and Mohamed et al.[17]). At the same time, increasing interest began to appear on what the design process means within the product life cycle, what critical decisions need to be made, and how experts approach the design tasks. Recently the interest increased, however the evaluation of an architectural design decision is characterized as a problem where solutions cannot really be false or true, at best they are good or bad.[18] Moreover, the inherent complexity and sporadic existence of empirical evidence in the software architecture decision-making process[19] justifiably created the need for analysis of decision cases,[20] human behavior and practice[5] and the quest for a unified language,[21] evidence-based knowledge repositories,[22] and evaluation of the assets composing the design.[23]

Considering systems built using existing software assets, a recent trend is the emergence of software ecosystems.[24] Manikas[25] and Wohlin et al.[26] conducted systematic literature reviews to provide an overview of the state of the area as well as to point to future research directions concerning software ecosystems, including decision-making for software ecosystems. The However, the research directions do not address the use of software assets coming from a software ecosystem, and in particular, the areas concerning decision-making related to developing composite software using software assets from a software ecosystem. An area closely related to software ecosystems is component-based software engineering, an overview of the research in this area can be found in Vale et al.[27] For example, Vale et al.[27] point to the need for more research in several areas, including empirical studies concerning the use and benefits of component-based software engineering in industry.

2.3 | Decision support for asset identification and selection

Research for supporting decisions for the selection of software assets has explored the state of practice and proposed as well as evaluated solutions for asset selection. We briefly give an overview of what has been done in the area, whereas more details are discussed in Sections 5 and 6, which focus on the specific research areas.

Exploratory studies investigated asset origin decision-making in practice based on a survey[28] and a case survey,[20] where a case survey is a survey of case studies.[29] Whereas these studies looked at sourcing options, studies also investigated decision-making at the component level. That is, a sourcing option (e.g., commercial COTS components) has been selected. In the next step, concrete vendors and components have been chosen. Li et al.[30] investigated the component level selection for commercial COTS components and Gerea[31] for open-source components.

In Badampudi et al.,[32] the process of decision-making for composite software architecture design is seen as a checklist, to support the selection between different origins of components, that is, in-house (developed or reused or inner source), open-source, COTS components, services or outsourced (including crowdsourcing). Alégroth et al.[33] build on the work by Badampudi et al.[32] when they proposed a decision process with different characteristics in relation to the original one by Badampudi et al.[32] and gathered feedback concerning the models with their proposed characteristics. Individual aspects within the decision-making processes have also been investigated, such as the properties that need to be considered when making decisions (cf. Chatzipetrou et al.[34,35] and Sentilles et al.[36]) or preference elicitation for trade-offs between properties (cf. Franke and Ciccozzi[37]).

Furthermore, to leverage on past evidence from decisions already made (e.g., within a company, or between collaborating companies sharing data) tool support has been proposed for storing both contextual and outcome data[38] and for implementing flexible decision processes, where a process can be created by invoking services.[39]

The roadmap presented here builds on the vision that software development will continue to move towards being composed of software assets, and that research and practice in software engineering will become more evidence-based.

The roadmap has a particular focus on decision-making for choosing asset origins in the intersection between composite software and evidence-based software engineering.

3 | METHODOLOGY

3.1 | Background

The roadmap is a result of a 5-year research with three partners with complementary expertise and 10–15 researchers active at any given point of time. The collaboration has been intensive during these five years of research, which was preceded by writing the application for research funding. The research has included many workshops and research meetings, as well as joint research studies, over the years. Moreover, a reference group has been connected to the project throughout its duration. The overall objective with the reference group has been to corroborate research novelty and industrial relevance. The reference group included experts from both academia and industry. The number of participants have varied somewhat over time, between 3–5 members, due to practical reasons, for example, people changing workplace. Meetings have been held on a yearly basis to obtain feedback on the research and advice on the continued research.

The researchers included in the project team all have PhD degrees except one, who during the project is conducting his research education. Half of his research education was conducted before joining the project and he is now in the final stages towards his PhD degree. Five of the authors are full professors. The median employment time in academia or at a research institute is 14 years, and the median industrial employment is 3 years. Moreover, the researchers have gained additional industrial experiences during their employment in academia when working in research projects in close collaboration with industry. For example, novel solutions have been developed together with industry, for example, Gorschek and Wohlin,[40] and also evaluated in industry, for example, Gorschek et al.[41] The researchers have complementary expertise in the area of the project, including topics such as software architecture, component-based software engineering, decision-making in software engineering and empirical studies.

The overall research question for the project was "How do we trade-off functionality, time to market, cost, quality and risk when developing competitive software-intensive systems using components and services?" Emphasis was placed on strategic, tactical and operational decision-making for composite software, as outlined in Section 1.

The project contributed towards this goal, but the research also led to the identification of many unsolved research challenges when trying to support the vision of the roadmap. Specifically, the research challenges emerged from three research studies: a systematic literature review,[42] a case survey,[20] and a survey.[28] The literature review revealed challenges through the identification of research gaps. The case survey revealed improvement areas during decision-making, indicating where the industry would benefit from research. The survey comprised questions dedicated to eliciting challenges. The methods and the results of the three studies, in particular in relation to the roadmap, are briefly summarized as follows:

- A systematic literature review on software component decision-making.[42] The systematic literature review was conducted following the snowballing guidelines by Wohlin.[43] The main objective of the review was to identify factors that could influence the decision to choose among different component origins and solutions. The purpose of the systematic review was also to identify research gaps and directions for future studies. For identification of relevant studies, two approaches were used. Two of the authors collaborated on the identification of studies through snowball sampling,[43] including both backward and forward snowballing. A start set of papers was identified to start the snowballing. Then, the third author, independently of the other authors, conducted a database search to complement the snowball search. Search strings were formulated for the database search. Inclusion and exclusion criteria were formulated and used in both searches. A data extraction form was designed to ensure a systematic and consistent extraction of the information from the papers included in the review. The quality of all papers included was assessed on the basis of rigor and relevance by using the guidelines by Ivarsson and Gorschek.[44]
  The study only identified a small set of papers (26) investigating the choice of component origins. A small subset (nine papers) was of empirical nature, using methods such as surveys, interview studies and case studies. This showed the need for further empirical investigations. Concerning factors influencing decisions 25 factors were identified. They were categorized in high-level themes (project metrics factors, external factors, and software development activity factors) and themes (e.g., project metrics factors were time, cost, effort, and quality). With regard to solutions, the focus was on proposing mathematical models focusing on calculating the optimal solution. The study also led to the identification of central research areas needed for decision-making, namely decision processes/models and asset properties (in the text above referred to as factors).
- A case survey of state of practice for choosing component origins.[20] The case survey was based on the guidelines by Larsson.[29] The case survey investigated how component origins are chosen in practice. The focus was on investigating the options considered and chosen, the stakeholders involved in the decision, and the criteria significant for making the decision. In total, 22 industrial cases were elicited through interviews and industrial projects of ORION project members. The majority of the cases were from the automotive and the telecommunication domains. A data extraction scheme was defined to ensure comparability of the cases. The extraction scheme contained different information categories to be elicited, such as meta-information, case context, decision criteria, methods, outcomes, and results, as well as an evaluation and reflection on the decision and its impact. Open coding was used to analyze qualitative information. Descriptive statistics and odds ratio were used to analyze quantitative data. Overall, 22 cases were identified. With regard to architectural decision-making, it was found that decisions were prepared in a group, though individuals made the decision. Contrary to the solutions identified in the systematic review (Badampudi et al.[42]) decisions were made in discussions in an ad hoc manner. Furthermore, the trade-offs made between factors in decision-making deviated from the trade-offs made in the literature review by Badampudi et al.[42] The decision outcome was evaluated retrospectively and it was found that only seven of the decisions of 22 were evaluated positively.
- A survey of state of practice concerning make-or-buy decisions.[28] The survey followed the guidelines by Kitchenham and Pfleeger.[45] The survey's goals were very similar to the case survey,[20] looking at options considered, decision processes, and criteria. Data was collected using a

cross-sectional web-based survey.[45] Responses were obtained using convenience sampling through personal contact networks, as well as social networks, such as Twitter and Facebook. In total, 152 complete responses were obtained. The survey included a total of 25 questions, including open-ended as well as free-text questions. Both demographic information and questions related to the individual goals of the survey were included. The survey was piloted with 15 pilot respondents. The survey used descriptive statistics and statistical tests for hypotheses testing. Qualitative information was systematically coded. The coding was validated, involving several researchers.

The result of the survey corresponds with the case survey[20] concerning expert judgment being the main method for decision-making. Borg et al.'s study showed, however, that decision process/models in the industry varied. Both systematic processes and ad hoc processes were found. The most generic factors in the decision were functionality and reliability. The study by Borg et al.[28] highlighted the importance of considering decision context and thus tailor decision-making methods to specific needs.

Furthermore, the research has been focused on developing and proposing solutions, such as processes (e.g., Badampudi et al.[32] and Alégroth et al.[33]), a taxonomy (e.g., Papatheocharous et al.[21]), methods, models and tools (e.g., Axelsson et al.,[39] Sentilles et al.,[36] Papatheocharous et al.,[46] and Cicchetti et al.[38]), as well as their empirical evaluation (e.g., Olsson et al.[47]). In addition to the three studies described above (cf. Badampudi et al.,[42] Petersen et al.,[20] and Borg et al.[28]), the results of the research focused on developing, proposing and evaluating solutions formed our views of the challenges within the areas. Table 1 shows the studies and their methods that were important with regard to the areas identified (see Section 4 for a definition of the areas).

3.2 | Process of roadmap construction

The development of the roadmap has been a 5-year process. During the course of the research project the areas emerged, and hence they were identified prior to defining them for the roadmap. The collaboration has resulted in a mutual understanding and terminology of the research area, including creating a joint taxonomy for the area[21] and a canvas for its usage.[20,46] Hence, the concrete activities resulting in the roadmap is the closure of the project and documentation of the further research needed in the area. In particular, the systematic literature review,[50] the case survey,[20] and the survey[28] provided essential input to formulate the roadmap. For the specific activities concerning the development of the roadmap, we followed a systematic process. The research process comprised the following steps:

1. Definition of research areas.
2. Formulation of research challenges for each area.
3. Reviews and updates of the research areas.

Definition of research areas: A meeting was held, based on the results from the research project, to define research areas needing further research within the context of evidence-based decision-making for composite software. The need for further research has, to a large extent, emerged as part of the project, although the research needed had not been structured in detail into concrete research areas. Thus, the discussion was focused on how to define the research areas, naming of the areas and how to delimit them from each other. Based on the discussion, a set of

TABLE 1 Methods and studies in the project in relation to areas in need of further research

Areas	Methods used in studies
Composite software architectures	Interviews and workshops with industry practitioners (Svahnberg and Gorschek[1])
Unified language and taxonomy	Taxonomy construction method[48] used in Papatheocharous et al.[21]
Decision processes and models	Multiple industrial case studies (Badampudi et al.[32]) and industrial web- based survey (Alégroth et al.[33])
Asset properties	Literature review (Sentilles et al.[36])
Decision trade-off	Industrial survey (Chatzipetrou et al.[34] additional data analysis on data collected for Borg et al.[28]) and quasi-experiment with students and practitioners (Franke and Ciccozzi[37])
Decision context	Literature review (Carlson et al.[49])
Evidence-based knowledge repository	Demonstrator implementation (Cicchetti et al.[38])
Aligning software architecture and decision-making	No studies in addition to the three studies above[20,28,42] in the project.
Sharing experiences and data	No studies in addition to the three studies above[20,28,42] in the project.
Useful evidence	No studies in addition to the three studies above[20,28,42] in the project.
Automated support	Demonstrator implementation (Cichetti et al.[38] and Axelsson et al.[39])

areas was formulated and circulated among the team members. Given that the areas were already identified during the course of the project, only a couple of iterations were needed to define and delimit a final set of areas. The iterations were done via e-mail. Three researchers coordinated and reviewed all material. Others also received the descriptions of the areas, but we did not require everybody to provide feedback on all descriptions. In addition, for each area we had an author, a contributor (second author) and a dedicated reviewer. As a result, short descriptions of each area were formulated and agreed upon.

Formulation of research challenges for each area: As mentioned above, for each area, a main driver and author, with expertise in the area, was identified and assigned responsibility for drafting a first version of the research challenges in each area. A second person was based on expertise and interest assigned the role of the secondary contributor to the area and also to be the primary reviewer. Instructions for the texts for the areas were communicated to all authors concerning both formal issues, such as length and number of references, and more style-oriented issues, including more focus on challenges than the background. To support the authors, an example of a area description of similar length as targeted was communicated to all authors.

Reviews and updates of the research areas: The roadmap work was coordinated by three researchers, one from each project partner. The area descriptions were communicated to them for review and coordination concerning content, style and references. The areas were reviewed, and updates were made. The other sections were written based on the area descriptions and the overall goal of the roadmap. The roadmap was then made available to all authors for review and revision. Finally, the close to final version of the roadmap was discussed at a half-day session at a second workshop. Based on the discussions, the roadmap was updated.

3.3 | Validity threats

The main validity threats (cf. Petersen and Gencel[51] for an overview and definition of validity threats in software engineering) for this research roadmap concern generalizability (transferability of the within and across different groups) and interpretive validity (objective interpretation of the findings).

Generalizability: The project focused on specific types of systems (software-intensive systems, cyber-physical systems), which are prevalent in domains such as automotive and telecommunication. Within these contexts data from industrial contexts (see Section 3.1 and Table 1) has been collected. However, the results and recommendations may vary for other system types (e.g., information systems and gaming). Results generated in academic contexts are often considered to be not transferable, as they were not based on industrial problems.[52] Given the industrial focus of the research and the authors' experience from working in the industry and conducting collaborative projects with companies, the threat is considered to be under control.

Interpretive validity: When defining a roadmap, there is always the risk that individual researchers' views are biased. To reduce the threat, researchers from different partners in the projects formulated the areas with the roles the main driver and author, a secondary contributor, and a reviewer (see Section 3.2). After that, the document was reviewed by all authors. Although involving multiple people may not rule out bias, the threat of individual persons' biases is reduced.

4 | AREAS OF RELEVANCE

This roadmap discusses challenges and research directions characterizing the topic of evidence-based decision-making for composite software. The initial discussion session and the following discussions, described in Section 3, divided the challenges into two main categories. The first category is concerned with challenges related to areas in need of research to enable evidence-based decision-making, and the second category relates to supporting the decision-making.

4.1 | Enabling decision-making

Figure 1 presents the areas (numbered classes in the figure) and their relations (associations with reading directions) in the form of a UML-diagram. Please also note the comments associated with the classes "Taxonomy," "Evidence-based knowledge repository," and "Decision-maker."

To enable decision-making for composite software, we need first of all an architecture supporting composition ❶. As decisions for composite software are made (e.g., choosing a specific COTS component), the software architecture may evolve. This is expressed through the association between the classes "Evidence-based decision" and "Composite software architecture." Furthermore, a shared vocabulary is essential to ensure a common understanding, both in the context of the decision as such, but also more broadly if we are going to use evidence from other cases (taxonomy ❷). The rationale for including the taxonomy was that, without a common vocabulary, there is a risk for misunderstandings

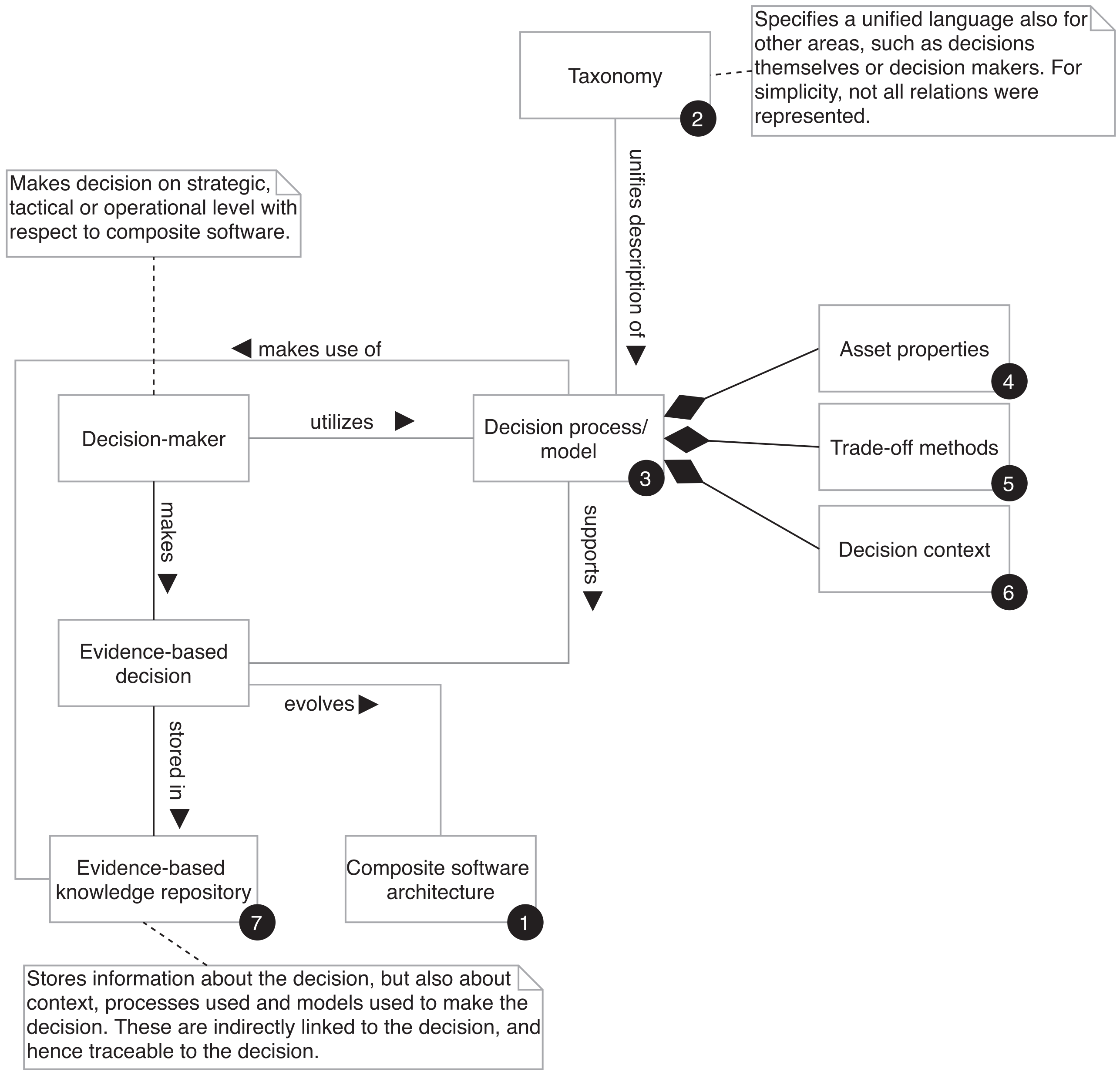
Taxonomy
2
Specifies a unified language also for other areas, such as decisions themselves or decision makers. For simplicity, not all relations were represented.
Makes decision on strategic, tactical or operational level with respect to composite software.
unifies description of ▼
◀ makes use of
Decision-maker
utilizes ▶
Decision process/ model
3
Asset properties
4
Trade-off methods
5
Decision context
6
makes ▼
supports ▼
Evidence-based decision
evolves ▶
stored in ▼
Evidence-based knowledge repository
7
Composite software architecture
1
Stores information about the decision, but also about context, processes used and models used to make the decision. These are indirectly linked to the decision, and hence traceable to the decision.

FIGURE 1 Overview of the research areas (enabling)

and evidence will be very hard to reuse between different decisions independent of the source of the evidence, which was highlighted by Papatheocharous et al.[21]

Some additional areas have been identified that are in need of research. A decision process or model ❸ is needed to support the decision-making to ensure that no aspects are forgotten and that no key stakeholders are left out. The decision process/model is also important to include as an area given that different processes/models vary in their performance (e.g., speed of decision-making, and flexibility).[33] Furthermore, a way of capturing different properties of assets ❹ being evaluated for potential use is required to ensure a systematic and evidence-based approach to decision-making. Looking at existing solutions for the choice of assets (see, e.g., the optimization models identified by Badampudi et al.[42]), and industrial practice[20,28] asset properties were key drivers in the decision-making, by which decision options are assessed. Therefore, they needed to be included as an area. Moreover, there is a need in relation to the decision process/model to have support for systematic trade-off analysis concerning different asset properties ❺, in particular, also related to the context ❻ of the decision. Given that not all desired asset properties may be achieved at the same time, there is a necessity to support making trade-offs,[53] which merits the inclusion of the area "Trade-off methods." The latter stresses the need also to have support for describing the context of the decision-making. It is needed to support an evidence-based approach, since evidence may very well be context-dependent. The importance of contextualizing phenomena in software engineering considering evidence was highlighted by various researchers (see, e.g., Petersen and Wohlin,[54] Briand et al.,[55] and Dybå[56]).

To support an evidence-based approach, there is a need for a knowledge repository ❼ where information can be stored and made available to decision-makers. The idea of an evidence-based knowledge repository has, for example, been highlighted in health care.[57] Information may be, for example, data, experiences and context descriptions concerning the decision made. When using the information stored, it is up to the user to decide if the information constitute credible evidence in relation to the case of the user. The knowledge repository needs to be populated with relevant information for decision-makers, which is challenging since there is a lack of knowledge concerning which knowledge is most critical.

The reflections above lead to seven areas of interest concerning the first category (enabling decision-making). The enumeration maps to the classes shown in Figure 1:

1. **Composite software architectures.** Developing decision-making suitable for a large variety of architectural styles and various application domains.
2. **Unified language and taxonomy.** Having a common language and structure to express and support decision-making.
3. **Decision processes and models.** Developing decision processes and models as checklists, to support the selection between different origins of assets.
4. **Asset properties.** Making possible the identification of the most important properties that different assets and solutions proposed bring.
5. **Decision trade-off.** Making decision trade-off scenarios for comparison and software asset selection by considering aspects such as asset properties and environmental/contextual factors.
6. **Decision context.** Using context to improve decision-making, as the suitability of solutions is highly dependent on the context.
7. **Evidence-based knowledge repository.** Designing a decision support knowledge repository balancing usefulness and documentation costs.

These seven areas and their challenges and research directions are addressed in Sections 5.1–5.7.

4.2 | Supporting decision-making

As mentioned, the second category of areas relates to the support needed in addition to the enabling areas for a systematic and evidence-based approach to decision-making for composite software.

First, it is essential to ensure that composite software architecture ❸ and decision-making supported by processes and models ❶ go hand in hand, in particular as support for architecting and technology continuously evolve. Furthermore, the decision space needs to be understood in terms of alternatives being available to software architects and other relevant stakeholders in the decision-making process today. Thus, the step of identifying alternatives to investigate is a central activity in decision-making processes (cf. Badampudi et al.[32] and Alégroth et al.[33])

Moreover, once a knowledge repository ❼ is populated with relevant information concerning decision-making for composite software, sharing data is a way to learn from company internal as well as external experiences. As Evidence-based Software Engineering (EBSE) is inspired by medicine, it is also interesting to observe that medicine strives for rapid dissemination and sharing of evidence[58] (e.g., in the form of rapid systematic reviews and their updates), but also discusses knowledge repositories.[57]

There is a need to be able to extract the most relevant and useful information from the repository ❼ given the context for the person looking for evidence (see Decision-maker in Figure 1). The idea of filtering information is central to EBSE, where the search for evidence and its critical appraisal are steps of the EBSE process.[9]

Finally, the decision-making process ❸ needs to be supported with tools so that as much as possible is automated, although primarily to support the decision-makers and not to make decisions for them. We argue that automation needs to be emphasized as soon as the data volume grows and many decision parameters (e.g., number of decision criteria, and contextual factors) are to be considered.

This leads to four areas of interest concerning the second category:

1. **Aligning software architecture and decision-making.** Identifying the most challenging architectural decision-making topics connected to architectural decisions.
2. **Sharing experiences and data**. Making use of competencies and experiences outside organizations, through ecosystems thinking and open innovation.
3. **Useful evidence.** Making sure research uses evidence-based practical knowledge to be applicable and relevant in practice.
4. **Automated support.** Aiding decision-making through automated tasks, frameworks and tools.

These four areas and their challenges and research directions are addressed in Sections 6.1–6.4.

5 | ASSET DECISION-MAKING CHALLENGES

5.1 | Composite software architectures

Composing a software system is easier today than ever before. Package managers are integrated into the development environments, and for many programming languages the package managers are linked to component repositories with a large selection of ready-made components, covering anything from simple helper functions to large and complex frameworks. For almost any task, there is a large variety of ready-made software assets available. However, challenges arise because of this ease of composition and availability of reusable assets including challenges in assessing candidate assets, challenges in understanding and resolving conflicting assumptions and dependencies, challenges in defining policies for asset composition, and challenges in providing adequate support for adhering to these policies.

The technology to find and access assets may differ significantly (e.g., downloading ready-made packages, cloning configuration management repositories, using language-specific package managers, or accessing the asset as an external service), and the licensing terms and responsiveness of the developing organization[1,34] influence how an asset may be used, updated, tested, and deployed. The component repositories may also be overpopulated with components that address similar issues, sometimes in conflicting ways.

Standalone assets in a system (e.g., databases, load balancers, web servers and monitoring software) are already either embedded into deployable units together with system assets (so-called microservices) or are offered and used as online services (e.g., authentication, or machine learning algorithms). Continuing this trend, entire applications are beginning to be developed as connected networks of microservices (partly to localize the effects of inter-dependencies between the compositional software assets) where some parts are developed or at least deployed in-house, and other parts are used as commodities "in the cloud." The challenges when developing with microservices are in many ways analogous to monolithic system construction. Managing versions and dependencies between packages and modules is equally important inside and between microservices, access and communication between microservices require equally standardized and secure communication flows and protocols,[59] and deciding how to complement the functionality provided by software assets (or microservices) with in-house developed wrapping code is equally challenging.

The evolving and changing nature of data and system configurations means that software architectures become the sum of all the decisions that define the parameters of a software and data architecture. Composite software architectures will thus define assets and connectors to only a limited extent, and mostly focus on defining policies for how to configure and connect assets, which assets and which data are permitted, where to find permissible assets, and within which constraints the system and the data must operate.[60,61]

Based on the above, the following main research directions have been identified:

- Support and research is needed to identify means and methods for developers to assess and select components according to a company's predefined criteria, and to communicate and coordinate these choices between all developers.
- Early analysis and visualization of the explicit and implicit dependencies (and other assumptions, e.g., about data formats and APIs) between software assets and their communication paths, and any therein identified security or other quality issues, remain a research challenge. Microservices add the complexity of doing this at runtime.
- Research is needed on identifying which policies are required, when and how to decide on the policies, and how to support the developers in adhering to the decided policies.

In summary, some architectural concepts such as namespaces and packages have already been internalized in the programming languages.[62] Internalizing support for the aforementioned challenges of policy definition and adherence, provisioning, testing, orchestration, deployment, monitoring, and runtime reconfiguration into the development environments is a vital key to solidify any research in any of these areas into state of practice. Increased support for automating this configuration[61] is expected and also a move towards automated visualization of architectural change. Specifically, microservices provide flexibility, but at the same time pose various challenges that need to be addressed. For example, with an increased number of services and deployment pipelines widely used automation servers, such as Jenkins, reach their limits in industrial applications.[63] Hasselbring and Steinacker[63] highlight the importance of microservice-based applications not sharing resources for low coupling, which poses challenges on smartly integrating services towards the end-user, and how to enable the system to achieve an overall system goal without resource sharing. A first idea of addressing the challenges is presented based on an implementation at Otto (cf. Hasselbring and Steinacker[63]).

5.2 | Unified language and taxonomy

A critical step in research is to ensure that collaborating researchers, as well as practitioners, in an area actively establish a common understanding of the field of study. Furthermore, this extends to the surrounding research community at large, where clear and consistent use of terms and concepts are critical when communicating findings. A common practice to enable these steps is the development and communication of a taxonomy. Creating taxonomies of objects or concepts is a common practice to "structure" a field or area of study (cf. Glass and Vessey,[64] Blum,[65] and Unterkalmsteiner et al.[66]). The most common example known in software engineering is SWEBOK.[67] A taxonomy gives structure, and the ability to communicate research results, and offers a base for theory building.[68] The development and evolution of taxonomies are essential to the accumulation of knowledge in software engineering.

The definition of taxonomies is also essential in evidence-based decision-making for composite software. We discuss the importance of context (Section 5.6) and an evidence-based knowledge repository (Section 5.7). Understanding the information provided by them requires definitions of the concepts and structuring the information, both essential steps for the process of creating taxonomies (cf. Usman et al.[69]).

It should be noted that the development of taxonomies is challenging. For example, there is a need to provide definitions for the elements of a taxonomy and assign them to categories, which also need to be described and defined.[69] The following reasons make the development of taxonomies challenging:

- **Deciding on the right level of detail and type of taxonomy:** A taxonomy that is too detailed and complex makes it hard to use. A too abstract taxonomy may lead to ambiguities. Also, the usage context may influence the right level of abstraction. Furthermore, there is a need to decide on the right type of taxonomy for the problem to solve (e.g., faceted versus hierarchical). In the context of decision-making for composite software, we do not know what the right level of abstraction is and which type of taxonomy is most suitable to support decision-makers.
- **Agreeing on terminology:** Within organizations, we observed that terms are also not understood in the same way. In different organizations, the understanding of what specific terms mean may vary even more. The reason may be that software engineering is a young research field, and thus, the community has not agreed on terminology. Companies also tend to develop their own company-specific "language," which makes knowledge transfer between organizations difficult.

To support decision-making for composite software, we created the GRADE taxonomy,[21] which structures information related to the goal, role, asset, decision, and environment. GRADE, like any taxonomy, can be seen as a type-definition language, that is, any decision to be taken within the area can be described using the base elements (with sub-elements) using the taxonomy. To address the two challenges mentioned above in the context of composite software, we propose the following directions for future research and also explain how they address the challenges above.

- **Design empirical studies of composite software using taxonomies:** Data collection instruments may be constructed based on taxonomies (such as GRADE). One example is the recent case survey on sourcing decisions for assets,[20] which used the taxonomy to identify relevant roles, assets concerned (e.g., components), decisions and contextual factors (environment). Specifically, it would be interesting to extend the study by Petersen et al.[20] collecting a large inventory of decision cases. The cases may be obtained from primary (e.g., collected through interviews) as well as secondary data sources (e.g., collected through systematic reviews). By using the taxonomy widely in the research community as a support during the design of studies, we will gain a shared understanding of the terminology (second challenge above).
- **Support industrial decision-making with taxonomies:** Future work should focus on integrating taxonomies, such as GRADE, in decision-making processes and tools. A shared understanding of terminology provided by taxonomies may be helpful when looking for evidence-based on past cases (see also Section 5.7). The initial version of GRADE was designed to be exhaustive. When using a taxonomy such as GRADE in practice, we will learn which information is essential to be captured and why (Challenge 1 above). At the same time a taxonomy helps us to communicate our knowledge about decision cases within an organization, or across organizational boundaries. Thus, a shared understanding may be achieved, leading to an agreed terminology (second challenge a).
- **Updating the taxonomy:** Conducting the studies suggested above will lead to revisions and improvements of taxonomies such as GRADE. Thus, it is essential to update the taxonomy continuously. We propose to provide a versioned online version of the taxonomies and a change history, including rationales for changes. We also suggest following the guidelines by Usman et al.[69] for developing taxonomies.

5.3 | Decision processes and models

Software development is decision intensive with strategic, tactical and operational decisions occurring with different frequencies and scope. Many of these decisions require a decision process that involves roles, assets, contextual factors, properties and activities.[32] Decision processes are abstracted into decision models that highlight the most important elements and actions for a decision.[70] The dynamic nature of software business and frequently changing customer needs require flexibility in decision processes and cause one single decision model to be infeasible for general use. Many decisions can be made following a simple model, but only some require a rigid and fully traceable process with documented justifications for decisions at each step. Simple decision models are easier to understand and to apply in an industrial setting and can give faster results by allowing for early evaluation of the decision (also called "fail fast"). The demands for simplicity and increased efficiency of decisions leads to the following challenges when developing decision models:

- **Speed**—day-to-day decisions are often made quickly requiring a decision model with high "speed" to arrive at a "good enough" decision. This implies limited investigation effort and often selecting the first promising option.
- **Flexibility**—decision models should provide value for decision-makers involved in both day-to-day decisions and large strategic investigations. Due to the high variety and number of internal variables, a high-level model is perceived to be required, but such a model will only provide limited support and value to its users. Balancing the abstraction level and utility remains a challenge to be addressed.
- **Agility**—an optimal decision model should support agile transformation and offer flexibility in adaptation to various organizations. Since organizations tend to implement agile methodologies in different ways, the most optimal decision model has to offer flexibility and adaptability. Agility also means decisions are made based on incomplete information rather than waiting for extensive analysis results.
- **Precision**—decision models that are flexible, fast, and agile could offer a "good enough" recommendation with a certain variance of the decision outcome in comparison to the optimal solution. For the decision scenarios with limited time and resources, this should be considered sufficient.
- **Fail fast**—the above listed four properties point towards a decision model that suggests the first feasible "good enough" option, or satisficing alternative as expressed by,[71] and let the company fail fast and learn the consequences. The process of following the first option quickly and learning along the way could be considered a part of the decision model where "bad solutions" are excluded and lead to refinement of the decision criteria, see Figure 2.

Another aspect to consider is the elements of these models, including the process, trade-off methods, and supporting repositories. The reason is that associated research has identified that decision-making in practice seldom follows strict decision processes or models.[72] A strict model, with many elements, could thereby be difficult for practitioners to adopt due to misalignment with current ways of working and thereby be rejected. Research should thereby not only aim to find new models but also study how to best integrate them into the current industrial environment. This requires research into model acceptance as well as incremental adoption processes. Finally, and perhaps most importantly, the decision outcomes of any proposed model needs to be compared to existing ways of working to evaluate their efficiency and effectiveness. Research results of this kind can help identify the critical parts of a model and what model elements that provide excess value versus cost. Thereby supplying evidence to encourage the adoption of more stringent models.

The main research direction that we would recommend is to focus on understanding the contextual ramifications and constraints that determine what decision model (simple or more complex) should be applied and how quickly that model should deliver a recommendation or suggestion. After understanding the context of a decision,[73] its criteria and the trade-offs between these criteria, decision-makers can apply a particular decision model or focus on collecting more evidence from the repository or from experts regarding the properties. This research thereby includes definition of different decision model types, or archetypes of decisions, for various situations and contexts. Thus, providing the practitioner with a toolbox of different models rather than one model that should fit all circumstances.

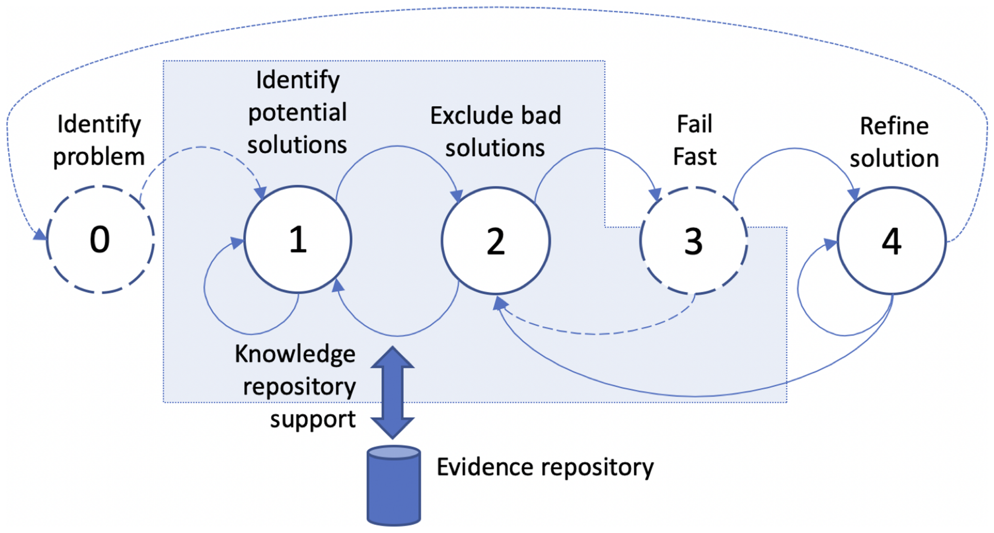

FIGURE 2 A simple decision model with five stages with early exclusion of solutions that are not suitable for the identified problem. The colored area highlights the stages supported by the evidence repository

Regardless of approach, any future decision modeling requires a strong industrial base to stand on, which implies that the research should be conducted in close collaboration with industry and focus on incremental improvements of state of practice rather than introducing a new set of practices. Although the latter may be more impactful, the former is more likely, from our experience, to succeed.

5.4 | Asset properties

In the decision-making process, it is important to identify the assets which are available and, then, to compare them according to several selected properties to choose the ones that best satisfy the expected goals. The comparison is grounded in the use of evaluation methods to quantitatively or qualitatively quantify the asset's properties summarizing the functional and non-functional characteristics of the asset. A functional property describes what the asset does and a non-functional property represents how well it is done.

For example, a typical decision-making process for software-intensive systems could include the following scenario: "Is it most cost-effective for our company to buy software component CPT1 from vendor V or should we develop the functionality ourselves as an in-house component (CPT2)?" In this scenario, two assets are weighed: a COTS component (CPT1) versus an in-house component (CPT2). For the decision, both components need to be evaluated according to their respective costs. Several properties directly linked to costs are, thus, relevant such as one-time price (CPT1), licensing fee (CPT1), development costs (CPT2), testing costs (CPT1, CPT2), integration costs (CPT1, CPT2), and maintenance costs (CPT1, CPT2). In addition, other properties, not related to costs, might also be worth taking into consideration in the comparison: functionality, timing performance, usability, security, reliability, and so forth.

Several classifications currently exist to help in the identification of the properties. The most predominant ones are those described in the ISO/IEC 9126 standard[74] and its successor ISO/IEC 25010 (SQUARE).[75] However, these classifications do not provide a list of evaluation methods to assess the corresponding properties. Furthermore, as demonstrated by several studies (cf. Adewumi et al.[76] and Al-Badareen et al.[77]), no classification supports all existing properties. This leads to a significant gap between research and practice and forces decision-makers to go through the state of the art to identify relevant properties and evaluation methods or blindly reuse methods and properties that have been used in previous projects.

Due to the lack of adequate classifications, the terminology is often inconsistently and loosely used. Different research communities and companies use different terms to refer to similar concepts as also discussed in Section 5.2. Conversely, the same terms can be used for different concepts. This results in the existence of synonymous, paronymous (having a similar form, but not the same), and polysemous (having multiple meanings) definitions, which make selecting properties and their evaluation methods even more challenging.

An additional challenge is incomplete and inconsistent specifications, including specifications of different properties. The methods used to quantify a property may not explicitly specify what they measure and how. For example, measuring software size can be done by using either the physical Source Lines of Code (SLOC) or the logical SLOC. SLOC does not have a clear definition, and in addition, depending on the use they might include comments and bracket lines, or not.

Furthermore, there are many ill-adapted and invalid metrics. Metrics are used to qualitatively or quantitatively quantify the properties. However, according to Kitchenham,[78] many of the commonly used metrics are not suitable or even wrong because they do not satisfy basic requirements for metrics as defined in measurement theory, they have not been validated, they do not necessarily measure what they should, and they do not provide a sufficient and necessary criterion to assess a property. At best, they can only serve as an indication. In spite of this, they are still widely used.

This implies that researchers need to better investigate how to tame the current state of the art on software quality. In that context, we identify the following as key future research directions in asset quality:

- **Increased use of evidence-based studies:** Systematic literature reviews, mapping studies, and large-scale surveys are needed defining suitable properties and useful evaluation methods based on evidence-based studies. That is, there is a need for reviewing property definitions in general, and to investigate how to use and evaluate the properties in the context of decision-making.
- **Development of a wide variety of open-source benchmarks:** Common benchmarks are essential to enable meaningful and rigorous comparisons and validations of research contributions. In particular, this would enable identify limitations of many evaluation methods and their applicability in different domains.

5.5 | Decision trade-off

Software asset selection, regardless of asset origin, per definition requires comparison and trade-off between aspects such as asset attributes and environmental/contextual factors. However, this comparison is in many cases challenging because there are often many aspects to consider, raising questions about how, what, when, and why components should be compared?

The first step of the comparison is, of course, to find assets to compare that can potentially solve the stakeholders' needs. It implies that functionality of the asset is the first attribute to compare.[28] However, fit-for-purpose is not the primary attribute at this stage, rather, the cost of the asset is the primary concern (cf. Borg et al.[28] and Chatzipetrou et al.[34]). This cost includes the time and money required for the adoption and adaptation of the asset. In addition, the longevity and technical support for the asset need to be considered, that is, estimations of the maintenance and evolution costs of the asset. In particular, for third-party components, for example, open-source or COTS components, vendor or community support are key aspects to consider to ensure that the maintenance costs of the asset are manageable over time.

Furthermore, predicting asset longevity requires both a technical understanding of the asset as well as knowledge about the system in which the asset will be used. Thus, it puts requirements on both the stakeholder(s) conducting the evaluation of the asset and the ones making the decision. Failure to estimate the architectural impact of an asset will result in technical debt and subsequently increase the maintenance and development cost of the system over time.

Next, fit-for-purpose shall be evaluated and there are many methods available to this end, such as the ATAM method (Architecture Tradeoff Analysis Method)[79] or the MOT (Management of Technology) framework.[80] Both ATAM and MOT provide a set of aspects to consider and systematically break these aspects down into support metrics for decision-making. Similar support is provided by the GRADE taxonomy[21] that has been defined to explicitly support asset, and asset origin, selection. Albeit different, all support methods outline aspects to consider to integrate an asset into the architecture of the system. Hence, together with previous steps, they provide input to a baseline process consisting of

1. Identification of possible assets to compare in a trade-off evaluation.
2. The trade-off of the assets—properties based on one of the existing methods considering for each asset:
   a. **Needs/requirements**—What does the asset need to do?
   b. **Stakeholders**—Who needs the asset to do what it does?
   c. **Context**—Where/when and how long will the asset have to do what it does?
3. Choice of the most appropriate asset based on the objective results of the method used for the trade-off evaluation.

In summary, the following three research directions need further research:

- Research into asset selection in practice has shown that practitioners do not use formal decision methods.[72] Instead, expert judgment and more subjective methods are used. An important subject of research is therefore how to help practitioners perform more objective trade-off analysis with minimal impact on existing decision-making.
- Another future research direction includes how to utilize contextual factors to improve the choices made when using a flexible decision trade-off method and adapt it to context-dependent decisions. Such factors include, but are not limited to, organizational policies and stakeholder requirements. Finding how/if these factors weigh in on the final decision result could help identify better methods of how and what to trade-off in a decision to make objectively better decisions. These factors and their impact could, for example, be documented as a taxonomy.
- Finally, research is needed to also take the importance of a decision into account in trade-off methods, and they should scale accordingly. Hence, for an operational (day-to-day) decision, a lightweight process may be used, whereas for more tactical or strategic decisions a more rigorous method is likely more suitable. The obvious reason being the range of adverse effects resulting from an incorrect decision in terms of increased cost or lost time.

5.6 | Decision context

Context plays a central role in software engineering (SE) research and practice. In software engineering research, Briand et al.[55] argue that context is essential as the suitability of solutions is highly dependent on the context. In industrial practice, various methods rely on contextual information. The experience factory[81] stores contextual information by classifying and characterizing the projects (such as project size and application domain) for which experiences are stored to identify similar projects. Requirements processes consider the specification of system context (e.g., existing systems and environmental usage scenarios) in addition to the specification of system requirements.[82] In the examples above, contextual information serves one primary purpose: making decisions. The experience factory helps in deciding which good practices to use in a project—context information in the requirements process aids in deciding which requirements are valid. Also, architectural decisions are assessed and interpreted having contextual information in mind.

Three challenges with respect to decision context are proposed as directions for future research:

- **Determining relevant context information:** For architectural decisions (and any other software engineering related decision), we have to ask, which contextual information to collect. Figure 3 shows examples of potential contextual information that may be of relevance. The figure has

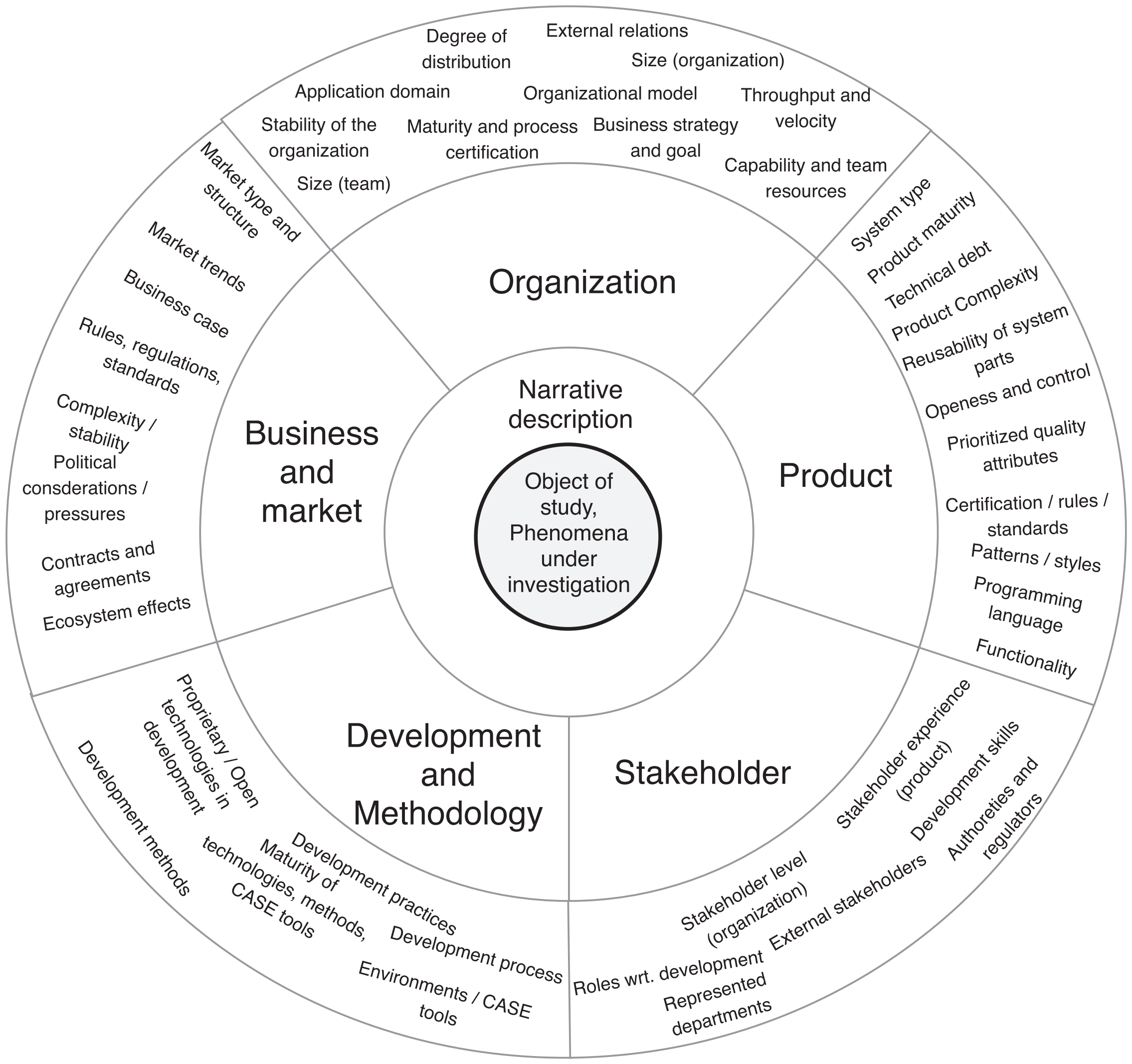
Degree of distribution
External relations
Size (organization)
Application domain
Organizational model
Throughput and velocity
Stability of the organization
Maturity and process certification
Business strategy and goal
Capability and team resources
Size (team)
Organization
Market type and structure
Market trends
Business case
Rules, regulations, standards
Complexity / stability
Political consderations / pressures
Contracts and agreements
Ecosystem effects
Business and market
Narrative description
Object of study, Phenomena under investigation
System type
Product maturity
Technical debt
Product Complexity
Reusability of system parts
Openess and control
Prioritized quality attributes
Certification / rules / standards
Patterns / styles
Programming language
Functionality
Product
Development and Methodology
Stakeholder
Proprietary / Open technologies in development
Development methods
Maturity of technologies, methods, CASE tools
Development practices
Development process
Environments / CASE tools
Stakeholder experience (product)
Development skills
Authoreties and regulators
Stakeholder level (organization)
External stakeholders
Roles wrt. development
Represented departments

FIGURE 3 Context information displaying five context dimensions

five dimensions: organization, product, stakeholder, business and market, and finally development and methodology). It also shows related context elements that may be suitable to describe, although it has to be decided from case to case which context elements are perceived as most relevant to capture. The context model, in Figure 3, is a visual representation of the model presented in Carlson et al.[49] To answer the question, we need empirical evidence based on a large set of architectural decisions. Specifically, software engineering research needs to investigate which context information plays an essential role in decision-making—recording all contextual information is infeasible. We may learn about this by associating the reported decision context with the decision outcome; for example, we learn that an architectural style is particularly well suited for software in a specific application type (e.g., context-embedded software). From a research method perspective, we suggest multiple directions: (a) gain in-depth insights of decisions through case study research and (b) gain general insights through a large-scale effort of collecting decision cases from practice through surveys and case surveys. The in-depth insights gained earlier may be used as a guide to developing surveys and case surveys. After having decided on the information to collect, there is a need to decide how to organize the information. Efforts have been made to classify contextual information (cf. Petersen and Wohlin[54] and Carlson et al.[49]), though further studies are needed, as researchers have not evaluated the classifications empirically yet.

- **Determining mechanisms to use context as a filter to identify relevant decisions:** Decision-makers would use context to find those decisions that are relevant to their problem. Thus, we need to investigate scenarios by which researchers would search for the decisions and also how to visualize the information so that it is specifically designed to support the identification and selection of relevant decisions. We propose to base the research on the insights obtained in the context of information visualization[83] to represent the mapping between context information used to search for decision cases and the search results. For example, we would like to learn how to present the information in such a way that an informed decision for the most relevant cases could be made.

- **Determining how to use contextual information in the context of decision-making:** Having found ways of presenting the information and identifying relevant cases, we need to investigate concrete decision-making processes in practice. In particular, it is worth investigating which context information to consider when looking for similar cases and how to weigh the different entries when evaluating similarities between cases.

5.7 | Evidence-based knowledge repository

Valuable outcomes from software architecting are not limited to the architecture specifications as such; instead, the importance of the decision process/reasoning behind design choices culminating in a software architecture is increasingly highlighted.[22] Indeed, the decision process and its decisions implicitly integrate domain-specific expertise and knowledge, which is useful to make future decisions in similar cases and for other related products. As a consequence, the research community is facing a growing need for storing architectural knowledge in an effective way. In fact, documentation of design decisions is, in general, a time-consuming activity that should show some payoff to be motivated (cf. Tang et al.[84] and Zdun et al.[85]). Designing a knowledge repository is far from being straightforward, at least for two main considerations: (i) the granularity of the decision details to be documented, and (ii) the data storage and maintenance.

Determining which level of decision detail to document: As mentioned before, any decision support system should provide a good balance between the effort required in storing and maintaining architectural details and the level of support in future architecting activities. Notably, the GRADE taxonomy provides a way to reason about decisions by means of a shared vocabulary of decision items and their properties.[21] Still, this kind of approaches can suffer subjective biases due to the involvement in a particular decision and the role of each decision stakeholder. Selecting and assigning priorities to decision goals are conditioned by the stakeholder's point of view (being a manager, an architect, a developer, etc.). Moreover, there exist decision details that are difficult to document because of their intangible nature, for example, perceived market trends, political relationships/partnerships, and so forth.

Another important issue to take into account is the amount of accumulated knowledge, which becomes critical when looking for evidence in stored decision cases. A repository with a limited amount of accumulated knowledge, that is, with very few decisions stored, would provide no reliable evidence to support future decisions. In this case, one possible solution could be the creation of a shared/public decision repository, in which companies would contribute with sanitized decision cases and would receive more reliable evidence for their future decisions as a payback. Needless to say, it would be of paramount importance to find a good balance between the usefulness of reusing decision knowledge and the cost and risks of publicly sharing internal decision information.

A possible mitigation for the problem of an insufficient knowledge base is artificial data generation. Decision cases can be automatically generated in a random but controlled way, in the sense that their elements can be randomly selected from a set of possible alternatives, but their structure has to be consistent with existing "real" cases (e.g., by obeying to a probability distribution). This approach could be easily extended to any company that owns a small number of cases and wants to get a larger knowledge base, however, the reliability of this solution in providing evidence for future decisions is an open research question.

Determining how to store and maintain decision data: A relevant corpus of literature has been devoted to investigating solutions for storing architectural knowledge.[86–88] Since one of the objectives of storing decisions is to provide support through evidence, the data storage should be conceived to allow the following: flexible data structures, since decisions might be documented at variable levels of detail; and multiple join queries in an acceptable computation time, due to the computation of similarity between decisions that necessarily combine multiple items of a certain decision (e.g., asset origins and decision stakeholders). By looking at other domains in which recommendation systems are highly exploited, graph structures appear to be an adequate solution. Indeed, these structures are malleable with respect to the variable granularity of the stored information. Moreover, graph structures work efficiently even when dealing with a huge amount of entries, for example, for social networks. Our research, in the above mentioned project (see Section 3), confirms the power and usability of graph structures. In particular, we concretely implemented a knowledge repository as a graph database through the Neo4j technology. Over such a database, we ran community detection algorithms to elicit families of decision cases and to possibly predict to which of those families a new case would belong.

A challenge due to the intrinsic nature of graph-based representations is that decision information tends to be semantically rich both on nodes and edges, whereas graph algorithms work on nodes and their interconnections. As a consequence, decision information needs to be "exploded/flattened" so that relevant details are all placed into nodes of the graph. In turn, this allows the graph algorithms to work in an optimal manner when eliciting similarities and communities.

Another interesting aspect related to decision support is that the outcome of a decision is rarely a binary value. As a matter of fact, the consequences of a decision are typically multifaceted, could trigger new decisions, and could even change over time. In this respect, an open challenge is representing the decision outcome(s) in the graph, and possibly maintaining the decision outcomes as time goes by—also considering the effort of documenting decisions in relation to its usefulness.

The challenges above all relate to research. However, the following two research directions are judged as particularly important to move forward in this area:,

- First, how to handle decision documentation biases, where the biases could affect the relevance of certain decision items, the elicitation/prioritization of properties that really affected a certain decision, and even the outcome of the decision itself. Without appropriate support in how to document decision cases, there is a risk that different individuals document them differently. Thus, decision cases will be difficult to compare and put into the context of the user. This implies that guidelines are needed to document decision cases to increase the likelihood that similar information is documented.
- The other area for which more investigation is demanded is how to measure the similarity between decision cases. Indeed, it is difficult to find investigations in existing software architecting literature that study what factors could be exploited to identify a certain decision case and its characteristics, as well as their weights. Having such factors at hand could allow a more reliable clustering of decision cases and retrieval of past cases for collecting evidence to support future decisions. If having guidelines as described in the previous research direction, the documentation of decision cases ought to become more alike. The ability to cluster similar decision cases is essential for users of a knowledge repository, since a cluster of similar cases may together be viewed as reliable evidence that may be used for future decision cases of similar nature. Thus, research into similarity analysis of decision cases and clustering technique are essential directions to move toward making more evidence-based decisions.

6 | SUPPORTING ASSET DECISION-MAKING

The areas described in Section 5 are primarily related to tangible descriptions and artifacts. The areas here are supporting activities to enable evidence-based decision-making for composite software. This also implies that the research directions identified in the sections below are of more general nature.

6.1 | Aligning software architecture and decision-making

The architecture of a (software) system is commonly defined as "the fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution."[89] Software architecture emerged in the 1990s as a discipline focusing on providing a high-level blueprint of a system. More recent work has emphasized the iterative nature of architecting, viewing it as a sequence of decisions.[90] This view is also ingrained in the current standard,[89] which emphasizes that architects should elicit stakeholder concerns, and make architectural decisions to address these. However, there is relatively little support existing for making such decisions, and it is often up to the knowledge and experience of the architect.

Many architects end up in one of a set of anti-patterns,[91] such as creating the perfect architecture for the wrong system or one that is too hard to implement; getting isolated from the rest of the organization in an "ivory tower"; or failing to make progress due to fire fighting. The result is an architecture description which does not relate to the real system, and as evolution continues it becomes more and more distant from reality. This indicates that current architectural decision-making methods do not adequately connect to developers and that better support would be needed to rapidly assess options. Also, mechanisms for enforcing, or at least monitoring adherence to, architectural decisions during development would be beneficial.

These problems are aggravated as agile practices become commonplace. There is an obvious conflict between the largely self-managed teams of agile versus the centralized, holistic decision-making in architecting, and between the emphasis on code in agile versus the document-centric top–down view often applied by architects. These issues are well known and have to some extent been studied in research.[92]

However, there are three more recent trends that are challenging to architectural decision-making:

- **Frameworks:** The view on pre-existing software components is changing. Often, the architecture is drawn up based on stakeholder concerns with a focus on functionality and life cycle. Eventually, the result is a set of interconnected elements, and those elements could be filled by relatively small COTS components. However, in modern software development, it is more common to instead base the system on a set of preselected frameworks. These are large and complex, and the developers adapt them to their needs by inserting additional code and parameters.
- **Systems-of-systems:** A software component is not necessarily integrated into the system, but a system-of-systems approach is increasingly common, where the system of interest relies on services provided by other systems, in a loosely coupled constellation. Examples of this can be found in software ecosystems, microservices, and so forth.
- **Continuous deployment:** The speed of evolution is increasing rapidly, and to handle this a continuous deployment approach has to be integrated into the system from the start. This intertwines the development tools with the actual system, making them hard to separate.

Techniques such as DevOps and the use of software containers are now commonplace and it would require operational monitoring of a running architecture.

These trends provide tremendous challenges to software architects and making the right decision in the selection of frameworks, strategy for interacting with external services, and providing a platform for evolution, can be decisive for the organization as a whole. It also shrinks the time window available for architectural decisions considerably.

We would, therefore, recommend further research into the following areas:

- Descriptions and analysis methods for frameworks and services that provide architects and developers with a better understanding of these fundamental concepts and allowing a rapid comparison and decision to support more continuous deployment.
- It is also difficult to estimate the effort of adapting frameworks and services, and current models of development effort prediction are focused on lines of codes produced, which is less relevant in this context.
- Challenges related to decision-making for systems-of-systems need further research. It includes decision-making and trade-offs concerning, for example, architectural choices in relation to properties such as safety and security when systems are interconnected.
- Finally, there is a need for better approaches to risk analysis connected to this kind of architectural decisions.

6.2 | Sharing experiences and data

Making decisions on whether to implement the software yourself, source the implementation from a contractor, buy a commercial component or to use an open-source component are frequently performed by software developers. However, the same decision for the same functionality is normally only done once! Software practitioners can benefit from from the experiences of others when developing new functionality. Learning from external experiences is highly valuable when developing new functionality or entering new domains or business contexts and can help to avoid some mistakes, errors and failures that others had to experience.

Hence, companies need to look outside their organization to complement their internal experience. The Open-Source Software movement and online discussion forums such as StackOverflow are two examples of how opening up to share experiences can benefit the communities of software practitioners and also society—similar to open innovation (cf. Chesbrough[93]). Runeson[94] suggests the Open Collaborative Data concept where data is shared in open collaboration between organizations, similar to OSS, which we believe is an important research direction also for sharing experience across organizations.

Before data and experience can be shared with others, it first needs to be described and represented in a manner others can interpret—for example, with a taxonomy such as GRADE[21]—and accurately described so others can relate to the content—specifically elaborating the context (cf. Briand et al.[55]). However, investing effort in documenting experiences and formatting data to share with others requires a mindset change and commitment from the company—similar to OSS (cf. Linåker et al.[95]). For a mindset change to happen, management goals need to be aligned with the personal goals of those responsible for the tasks. Furthermore, experience sharing needs to be seen as more than a purely technical matter to be successful; personal networks and social capital are essential to both novice and experienced developers.[96,97]

Organizations should create clear guidelines on when information or experiences can be shared with others as common knowledge.[95] We believe that any "company sensitive" knowledge or technology sooner or later become a commodity and therefore can be shared externally without the risk of losing valuable business assets—which is along the same lines of Runeson.[94] Hence, an interesting research topic is how to decide what to share and when.

We see a need for utilizing external information to be able to make informed decisions. By having an open innovation[93] mindset, crowdsourcing can offer a collective knowledge base,[62] where it is also often possible for individuals to develop their own network and reputation in a community besides getting access to the knowledge on platforms such as StackOverflow. However, tools might be needed to search the vast content as the platforms might lack a clear taxonomy and well-defined context description.

The research directions needing further research include

- We need to better understand what to share and when to share it to enable informed decision-making.
- In relation to the previous item, there is a need to understand how to document the context to make the information useful to others.
- Furthermore, research is needed concerning how to support decision sharing, for example, in terms of tools.

6.3 | Useful evidence

Evidence (for example in the form of systematic literature outcomes or from individual studies) should provide conclusions on a topic and recommendations that could be useful for making decisions about practice.[98] In architectural decisions, the evidence is useful to make informed decisions. For example, decision-makers can benefit by knowing the important properties in a specific situation, such as time, cost and quality when choosing among different development alternatives. Providing evidence on factors that are useful in making architectural decisions will support decision-makers in considering new factors they did not know and/or confirm the factors that are known to them.

The importance of using evidence in decision-making has been introduced in software engineering for researchers and practitioners.[10] Industry is showing a growing interest in evidence, that is, the outcomes of evidence-based software engineering research.[99] The outcomes of research should impact practice by providing pragmatic and actionable advice.[100] For example, evidence in the form of recommended criteria or decision-making processes to consider when choosing an OSS component is useful in decision-making.

However, there is often a gap between research and practice. For example, the criteria to select among different asset origins in practice are not always considered by solutions proposed in the literature.[20] Therefore, for the evidence to be useful it is important to assess if the evidence is applicable or relevant in practice. It is also important to provide evidence, that is, recommendations with different assumptions, levels of rigor and levels of confidence in the recommendations.[100] The details of the context in which the evidence was produced help to assess the usefulness of evidence in architectural decisions. For example, the decision-maker can assess if a particular property is applicable to their context based on the context in which the knowledge was produced, or get help to identify suitable methods to assess a specific property.

Given the interest and importance of applying research results in practice, the activity of knowledge translation (KT) has emerged as a research activity in software engineering.[99,101] Some of the steps in KT include finding relevant evidence, gathering the current knowledge of stakeholders in the context in which evidence is to be translated and and aligning evidence to context. The main goal of KT is to support the objective of evidence-based software engineering, which is to "to provide the means by which current best evidence from research can be integrated with practical experience."[10]

Recently, we can see an increase in research studies that provide support to the KT activity. For example,

- An approach to present the evidence in a practitioner-friendly format called evidence briefings is presented in Cartaxo et al.[102]
- A new way to label and organize evidence based on its varied quality makes the evidence available in a pragmatically useful form.[100]
- A synthesis method that takes into account both the practitioners' experience and evidence to contextualize the knowledge is presented in Badampudi and Wohlin.[50] The synthesis method by Badampudi and Wohlin[50] is based on Bayesian principles which start by collecting the prior opinion that is based on the experience of the practitioners which is then revised in the light of the evidence provided to the practitioners.

We see a paradigm shift towards evidence-based software engineering as researchers aim at supporting decisions with evidence for software engineering practice.[100] A system that allows researchers and practitioners to reliably synthesize research results into actionable, real-world guidance is envisioned as the next steps in software engineering research.[100] In the future, we may expect more studies focusing on the contextualization of evidence for making evidence-based architectural decisions. The relevance and applicability of evidence can be assessed through KT which will, in turn, provide an input to produce more contextualized and useful evidence. We believe the community of empirical research needs to move to a systematic way of producing contextualized evidence that is useful for making evidence-based architectural decision.

The following research directions exemplify some of the research needed:

- We need to find ways to identify relevant and valid evidence in relation to our context.
- Given that existing evidence may not fit perfectly with a different context, we need to research how experiences from one context may be useful in (or adapted to) a different context.

6.4 | Automated support

An approach aiming at supporting software and "soft" hardware engineering cannot take away the attention from automation.[103] In later stages of an engineering process, powerful mechanisms to automate tasks, such as code generation from design artifacts or testing, are inherently simpler to achieve than at earlier stages. The reason is that development artifacts become more and more detailed along the way and allow for precise and extensive manipulations and transpositions to automatically generate other artifacts or improve existing ones. In the early stages of development, reference artifacts, such as requirements but also architectural sketches and related documentation, can be more volatile,[104] and, hence, automation becomes harder to achieve.

Making decisions on how to architect software is a glaring example of such an early development stage. However, supporting decision-makers by leveraging relevant knowledge in an evidence-based fashion can be both highly rewarding and very challenging,[8] in particular for practitioners in real-life settings.[9] Providing decision support to decide on whether to go for COTS components or in-house developed components, for example, requires analyzing and presenting a combination of syntactical (i.e., what decisions have been made in similar settings in previous projects) and semantic (i.e., what the rationale behind the decision was) data to the decision-maker. Automating such a task is complex, but, if successful, could also be most valuable.

In automating software engineering aspects, it is usually complex to combine syntax and semantics for storing, retrieving and presenting data to the intended stakeholder. For combining syntax and semantics, several means can be used as a lever: (1) ontologies for the formalization of standard properties, (2) models and computational algorithms to assess software quality, (3) powerful graph databases and self-querying mechanisms to encode decision cases in graphs, and (4) graph analyses to identify relationships and balance between different architectural knowledge elements.

Although being able to achieve partial automation with the aforementioned tools, we can safely claim that a pervasive issue when dealing with automated support of decision-making processes is the formalization and manipulation of semantic knowledge alongside pure syntactical data. A large and representative set of decision cases, more or less detailed, becomes a solid ground for automating decision-making only if they are equipped with a malleable semantic rationale of how and why the specific decision was made. There is a substantial body of literature on the management of semantic knowledge, but most research to date has targeted the so-called semantic web.[105]

For decision-making, two primary research questions (directions) related to semantic knowledge of decision cases need to be answered and supported by tools:

- How to store semantic rationales of a decision case along with the corresponding data? We believe that ontology-based approaches should be empowered with powerful and flexible mechanisms to arrange syntactical information to implicitly delineate its underlying semantics. Graph-based solutions could be used here, but we need to make them able to represent semantics in a malleable manner.
- How to automatically retrieve and present the data of a decision case? Apart from gathering and storing data, its retrieval and proper presentation to the stakeholders are pivotal tasks. Current solutions do not discern between the type of stakeholder at hand, which is instead a major concern to properly support the decision-maker. In this direction, we envision multi-level support for isolating the expectations and wishes of diverse decision-makers (e.g., software architect vs. product owner) and, based on them, provide a tailored and customizable retrieval and presentation of relevant aspects of stored decision cases to support decision-making.

7 | DISCUSSION AND SUMMARY

The vision of software engineering being an evidence-based discipline comes with challenges. Most importantly, this includes the pace in which the area is evolving. New technologies, methodologies and tools are continuously introduced into software development. Furthermore, the size of current software systems and their creation by a large number of developers in teams often distributed across the globe make it even more challenging. Thus, we may not be able to use an evidence-based approach for all activities in software engineering, at least not yet. Thus, it is important to identify areas where an evidence-based approach may work. Such areas and cases may be forerunners for further areas to be evidence-based in software engineering.

In this paper, a roadmap for an area that may be a suitable forerunner has been presented. The area is suitable since decisions on how to select which parts need to be developed or reused in-house versus other options (including open-source software, COTS components or outsourcing) are independent of new technologies, methodologies and tools. Furthermore, such decisions are already now common in larger software development projects. Thus, the evidence does exist, although it is most often not documented. This means that there is a potential to make future decisions evidence-based if the decisions made are documented, whether being company internal or across companies. Unfortunately, it requires an investment to document decisions. There is a risk that such an investment is viewed in a similar way as documenting the design rationale for different solutions. It should be acknowledged that it may be a cost in the short term, but the objective is that it should be an investment for the future. As the complexity of the software grows and new people come into the development, such documentation may be worth the upfront cost. The intention is to avoid having to redo work that has been done in the past; that is, we ought to learn from past experiences.

7.1 | Reflections on research areas

A number of the building blocks needing more research to move towards the vision has been presented. Overall, there is a need for more flexible architectural models for composite software architectures to enable building and evolving the software systems of the future. To move towards evidence-based decision-making in the area, there is a need to have a common vocabulary to be able synthesize evidence from different research studies and make the collected evidence available without misunderstandings. Thus, a community agreed taxonomy is needed, which also should evolve with the area. The requirements on software qualities are important and hence there is a need to be able to assess and evaluate different properties of both software assets and the overall software system. Furthermore, it is hypothesized that some support artifacts, for example, processes and models, would be required to provide objective decision support for, in particular, more tactical or strategic decisions. These artifacts need to be flexible, since decisions are taken on different levels, and depending on the level and the type of decision, different support is required to support cost-efficient decision-making.

As part of the decision-making, there is a need to perform trade-offs, for example, different properties of software as well as surrounding factors such as, for example, licensing and life cycle aspects. Thus, further research is needed into if, and how, trade-off analysis can be made more cost-efficiently for the important properties for different software solutions. Moreover, decisions are made in context and hence research is needed into understanding which, if any, context factors are important in different situations and if these factors can be used to determine the representativeness, and thereby value, of a previous decision for a new decision. To support evidence-based decision-making, evidence needs to be stored and retrieved. This includes having a repository for evidence cases, and good ways of retrieving the most similar cases to support decision-making. A useful repository means that data has to be shared, and there is a need for research into which data to share including context for the evidence. The seven areas presented in Section 5 need research individually, but moreover, research is needed into how to fit the building blocks together to support industry to move towards evidence-based decision-making for composite software. The building blocks need to be well integrated into existing processes, methods, models and tools used at companies, and hence research in close industrial collaboration is needed.

Furthermore, research challenges concerning support for conducting evidence-based decision-making for composite software have been presented in Section 6. Research is needed into how to efficiently make decisions for different composite software architectures as well as researching which decisions are needed for this type of architecture. There is a need to relatively easy identify and use the data. Thus, research is needed concerning information retrieval of the most relevant evidence. Finally, the decision-making process, including the building blocks as well as the repository and the evidence, needs to be supported by automation.

Research is needed in all 11 areas identified in this roadmap. However, the areas may be divided into a priority order based on time. For example, it does not make sense to build support for evidence-based decision-making if we do not have the enablers for decision-making in place first. Thus, in general the areas presented in Section 5 are a prerequisite for being able to support decision-making as outlined in Section 6. This does not imply that the areas presented in Section 5 need to be fully in place before addressing the areas in Section 6. We would suggest prioritization as follows:

1. A unified language and a taxonomy are the starting points (Section 5.2). A common vocabulary is a prerequisite, preferably across organizations, but at least within each organization intending to practice evidence-based decision-making. Furthermore, unifying software architecture and decision-making is needed early to help guide the research (Section 6.1).
2. Second, the appropriate models need to be in place. This includes decision processes and models (Section 5.3), as well as describing asset properties (Section 5.4) consistently and to capture the context of the decisions consistently through context models (Section 5.6).
3. Once the processes and models are in place, it becomes possible to make decision trade-offs (Section 5.5).
4. The enablers need to support different types of architectures (Section 5.1), and the decisions and their context ought to be stored for systematic retrieval of previous cases (Section 5.7).
5. To be evidence-based means both sharing (Section 6.2) and using data (Section 6.3), both within and when possible across organizations.
6. Finally, automation is needed. On the one hand, automation requires that many building blocks are in place, and hence automation comes last in time order (Section 6.4). On the other hand, the full benefits of an evidence-based approach to decision-making requires automation.

Taken altogether, the 11 areas point to a number of important research challenges. However, the integration of them into a comprehensive evidence-based decision-making framework supported by automation, including, for example, machine learning, also requires further research. The integration is a challenge in itself. Moreover, it requires substantial collaboration between research and practice to make evidence-based software engineering come through, including evidence-based decision-making for composite software systems.

7.2 | Reflections on the vision

Dybå et al.[9] provided an overview of the evidence-based software engineering (EBSE) process and discussed how the steps could be used in software engineering. Their focus was on evidence provided in the peer-reviewed literature investigating evidence for answerable questions (e.g., related to specific interventions). When using the EBSE process to solve a concrete industrial problem, Kasoju et al.[106] tailored the process to find tangible improvements for an automotive test process. Specifically, Kasoju et al.[106] highlighted a need to look at many different software engineering areas to solve the problem, including testing, requirements, processes, and so forth. The challenge was to use evidence still while not answering a very specific and focused question (e.g., related to one intervention), as this would not have helped solve the overall problem. In this roadmap, we also looked at solving an industrial problem based on the ideas of EBSE, and observed that various aspects needed to be considered, as is evident by the range of areas identified.

Therefore, we reflect on how the areas relate to the steps of the EBSE process suggested by Dybå et al.[9] We first briefly summarize the EBSE steps in the following:

1. *Problem identification and question formulation:* An industrially relevant problem is identified. The problem is the basis for formulating questions that need to be answered to solve the problem.
2. *Search for evidence:* Evidence for answering the questions is identified, for example, by looking at peer-reviewed literature (such as journal articles and conference papers), or gray literature (such as blogs).
3. *Critical appraisal of the evidence:* The evidence identified in the search may not all be relevant or of sufficient validity, for example, depending on the level of scientific rigor.
4. *Make use of evidence in decisions:* The evidence needs to be put to use in a practical context. Dybå et al.[9] point out that the difficulty of applying the evidence depends on the scope of application. Dybå et al. provide the example of a method used by a single developer, where the application is easier compared to a larger software process initiative, as was discussed by Kasoju et al.[106]
5. *Reflect on the EBSE process:* In this step, a reflection of conducting the first four steps of the EBSE process takes place.

The research areas and their interactions support the different steps of the EBSE process as follows:

1. *Problem identification and question formulation:* The industrial challenge is to support decisions related to the usage of assets in *composite software architectures*. Our earlier studies (e.g., the case survey and survey reported in Section 3.1) provided an insight into the decisions made. For example, Petersen et al.[20] found that the most frequently considered options for asset sourcing were in-house, COTS, and outsourcing. The decision-makers drive the problem in Figure 2.
2. *Search for evidence:* In analogy to the search for evidence in EBSE (e.g., scientific databases), the evidence-based knowledge repository is the source in our context. The decision-makers search the repository for past decisions that may serve as relevant for the decision problem. For example, the decision-makers decide between in-house and COTS and find past decisions as input. Various research areas aid the search for evidence. The area concerning a *Unified language and taxonomy*eases the search for information as the terminology is well defined. The *Decision context* area helps find those past decisions closest to the new decision situation (e.g., concerning domain, and system complexity). The area for *Asset properties* helps find the past decisions related to the properties of interest in the new decision.
3. *Critical appraisal of the evidence*: The decisions are critically reviewed for relevance and validity. The need to invest in checking for relevance depends on the accuracy of the search using *decision context* and *asset properties*, as well as decision options considered for filtering. Validity relates to the credibility of the evidence, which is may be affected by the rigor of the decision process. We may trust decisions more if the way they were reached is traceable and credible.
4. *Make use of evidence in decisions:* In this step, the decision-makers decide what constitutes good enough evidence. It is an outcome of the appraisal in the previous step based on the information retrieved from the *evidence-based knowledge repository*.
5. *Reflect on the EBSE process:* In this final step, the decision-makers reflect on the process and the decision made in the previous step. Storing the insights gained in the new decision in the *evidence-based knowledge repository* facilitates learning.

Overall, the mapping of the areas to the EBSE process shows that the realization and integration of the areas may support the EBSE idea.

8 | CONCLUSION

Evidence-based software engineering is a challenge, in particular when it comes to making decisions in practice based on evidence from research. The pace of changes in the way software is developed inhibits, most likely, the introduction of an evidence-based approach concerning all areas of software engineering. However, it should be feasible for areas where, for example, the evidence is continuously generated through frequently occurring decisions in practice. The latter is the case for decisions concerning composite software; that is, decisions are made with respect to which types of software assets to use, for example, open-source software or COTS components, and which particular software asset to use, for example, which open-source component to integrate. Thus, the research challenges and directions highlighted in this roadmap point towards supporting evidence-based practice through research, as well as capturing the evidence available in the research literature, for example, through published case studies.

FUNDING INFORMATION

Knowledge Foundation, Sweden, Grant/Award Number: 20140218

ORCID

Kai Petersen https://orcid.org/0000-0002-1532-8223

REFERENCES

1. Svahnberg M, Gorschek T. A model for assessing and re-assessing the value of software reuse. *J Softw: Evol Process*. 2017;29(4):e1806.
2. Zannier C, Chiasson MW, Maurer F. A model of design decision making based on empirical results of interviews with software designers. *Inf Softw Technol*. 2007;49(6):637-653.
3. Rekha VS, Muccini H. Group decision-making in software architecture: a study on industrial practices. *Inf Softw Technol*. 2018;101:51-63.
4. Ruhe G. Intelligent support for selection of COTS products. In: Web, Web-Services, and Database Systems, Lecture Notes in Computer Science. Vol 2593. Springer; 2002:34-45.
5. van Vliet H, Tang A. Decision making in software architecture. *J Syst Softw*. 2016;117:638-644.
6. Aurum A, Wohlin C. The fundamental nature of requirements engineering activities as a decision-making process. *Inf Softw Technol*. 2003;45(14): 945-954.
7. Antony RN. *Planning and Control Systems: A Framework for Analysis*. Boston: Graduate School of Business Administration, Harvard University; 1965.
8. Kitchenham BA, Jørgensen M, Dybå T. Evidence-based software engineering. In: Proceedings of the 26th International Conference on Software Engineering. IEEE; 2004:273-281.
9. Dybå T, Kitchenham BA, Jørgensen M. Evidence-based software engineering for practitioners. *IEEE Softw*. 2005;22(1):58-65.
10. Kitchenham BA, Budgen D, Brereton P. *Evidence-Based Software Engineering and Systematic Reviews*. Vol 4. CRC press; 2015.
11. Nowak A, Schünemann HJ. Toward evidence-based software engineering: lessons learned in healthcare application development. *IEEE Softw*. 2017; 34(5):67-71.
12. Finkelstein A, ed. Proceedings of the 22nd International Conference on Software Engineering, Future of Software Engineering Track (ICSE 2000) ACM; 2000.
13. de Lemos R, Giese H, Müller HA, et al. Software engineering for self-adaptive systems: a second research roadmap. In: Software Engineering for Self-Adaptive Systems II: Springer; 2013:1-32.
14. Pretschner A, Broy M, Krüger IH, Stauner T. Software engineering for automotive systems: a roadmap. In: Proceedings of the International Conference on Software Engineering (ICSE 2007), workshop on the future of software engineering (FOSE 2007). IEEE Computer Society; 2007:55-71.
15. Nierstrasz O, Meijler TD. Research directions in software composition. *ACM Comput Surv*. 1995;27(2):262-264.
16. Shaw M, Garlan D. *Software Architecture: Perspectives on an Emerging Discipline*. USA: Prentice-Hall, Inc.; 1996.
17. Mohamed A, Ruhe G, Eberlein A. COTS selection: past, present, and future. In: Proceedings of the 14th Annual IEEE International Conference and Workshop on Engineering of Computer Based Systems (ECBS 2007). IEEE Computer Society; 2007:103-114.
18. van Vliet H. Software architecture knowledge management. In: Proceedings of the 19th Australian Software Engineering Conference (ASWEC 2008). IEEE Computer Society; 2008:24-31.
19. Razavian M, Paech B, Tang A. Empirical research for software architecture decision making: an analysis. *J Syst Softw*. 2019;149:360-381.
20. Petersen K, Badampudi D, Shah SMA, et al. Choosing component origins for software intensive systems: in-house, COTS, OSS or outsourcing?—a case survey. *IEEE Trans Softw Eng*. 2018;44(3):237-261.
21. Papatheocharous E, Wnuk K, Petersen K, et al. The GRADE taxonomy for supporting decision-making of asset selection in software-intensive system development. *Inf Softw Technol*. 2018;100:1-17.
22. Kruchten P, Lago P, Van Vliet H. Building up and reasoning about architectural knowledge. In: Proceedings of the International Conference on the Quality of Software Architectures Springer; 2006:43-58.
23. Clements P, Kazman R, Klein M. *Evaluating Software Architectures: Methods and Case Studies*: Addison Wesley; 2001.
24. Bosch J, Bosch-Sijtsema P. From integration to composition: on the impact of software product lines, global development and ecosystems. *J Syst Softw*. 2010;83(1):67-76.
25. Manikas K. Revisiting software ecosystems research: a longitudinal literature study. *J Syst Softw*. 2016;117:84-103.
26. Wohlin C, Mendes E, Felizardo K, Kalinowski M. Guidelines for the search strategy to update systematic literature reviews in software engineering. *J Inf Softw Technol*. 2020;127(111):106366.
27. Vale T, Crnkovic I, de Almeida ES, da Mota Silveira Neto PA, Cavalcanti YC, de Lemos Meira SR. Twenty-eight years of component-based software engineering. *J Syst Softw*. 2016;111:128-148.
28. Borg M, Chatzipetrou P, Wnuk K, et al. Selecting component sourcing options: a survey of software engineering's broader make-or-buy decisions. *Inf Softw Technol*. 2019;112:18-34.
29. Larsson R. Case survey methodology: quantitative analysis of patterns across case studies. *Acad Manag J*. 1993;36(6):1515-1546.
30. Li J, Conradi R, Bunse C, Torchiano M, Slyngstad OPN, Morisio M. Development with off-the-shelf components: 10 facts. *IEEE Softw*. 2009;26(2): 80-87.
31. Gerea MM. Selection of Open Source Components—A Qualitative Survey in Norwegian IT Industry. NTNU Master thesis, Institutt for datateknikk og informasjonsvitenskap, NTNU, Norway; 2007.
32. Badampudi D, Wnuk K, Wohlin C, Franke U, Smite D, Cicchetti A. A decision-making process-line for selection of software asset origins and components. *J Syst Softw*. 2018;135:88-104.
33. Alégroth E, Gorschek T, Petersen K, Mattsson M. Characteristics that affect preference of decision models for asset selection: an industrial questionnaire survey. *Softw Qual J*. 2020;28(4):1675-1707.
34. Chatzipetrou P, Alégroth E, Papatheocharous E, Borg M, Gorschek T, Wnuk K. Component selection in software engineering—which attributes are the most important in the decision process? In: Proceedings of the 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA 2018) IEEE; 2018:198-205.
35. Chatzipetrou P, Papatheocharous E, Wnuk K, Borg M, Alégroth E, Gorschek T. Component attributes and their importance in decisions and component selection. *Softw Qual J*. 2020;28(June):567-593.

36. Sentilles S, Papatheocharous E, Ciccozzi F, Petersen K. A property model ontology. In: Proceedings of the 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA 2016). IEEE Computer Society; 2016:165-172.
37. Franke U, Ciccozzi F. Characterization of trade-off preferences between non-functional properties. *Inf Syst*. 2018;74(May):86-102.
38. Cicchetti A, Borg M, Sentilles S, Wnuk K, Carlson J, Papatheocharous E. Towards software assets origin selection supported by a knowledge repository. In: Proceedings of the 1st International Workshop on Decision Making in Software Architecture (March 2016). IEEE; 2016:22-29.
39. Axelsson J, Franke U, Carlson J, Sentilles S, Cicchetti A. Towards the architecture of a decision support ecosystem for system component selection. In: Proceedings of the Annual IEEE International Systems Conference (SYSCON 2017). IEEE; 2017:1-7.
40. Gorschek T, Wohlin C. Requirements abstraction model. *Requir Eng*. 2006;11(1):79-101.
41. Gorschek T, Garre P, Larsson SBM, Wohlin C. Industry evaluation of the requirements abstraction model. *Requir Eng*. 2007;12(3):163-190.
42. Badampudi D, Wohlin C, Petersen K. Software component decision-making: in-house, OSS, COTS or outsourcing—a systematic literature review. *J Syst Softw*. 2016;121(11):105-124.
43. Wohlin C. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering. New York, NY, USA: ACM; 2014:1-10.
44. Ivarsson M, Gorschek T. A method for evaluating rigor and industrial relevance of technology evaluations. *Empir Softw Eng*. 2011;16(June):365-395.
45. Kitchenham BA, Pfleeger SL. Personal opinion surveys. In: Shull F, Singer J, Sjøberg DIK, eds. *Guide to Advanced Empirical Software Engineering*. Springer; 2008:63-92.
46. Papatheocharous E, Petersen K, Axelsson J, et al. The GRADE decision canvas for classification and reflection on architecture decisions. In: Proceedings of the 12th International Conference on Evaluation of Novel Approaches to Software Engineering; 2017:187-194.
47. Olsson T, Wnuk K, Gorschek T. An empirical study on decision making for quality requirements. *J Syst Softw*. 2019;149:217-233.
48. Bayona-Oré S, Calvo-Manzano JA, Cuevas G, San-Feliu T. Critical success factors taxonomy for software process deployment. *Softw Qual J*. 2014; 22(1):21-48.
49. Carlson J, Papatheocharous E, Petersen K. A context model for architectural decision support. In: Proceedings of the 1st International Workshop on Decision Making in Software Architecture (March 2016). IEEE; 2016:9-15.
50. Badampudi D, Wohlin C. Bayesian synthesis for knowledge translation in software engineering: method and illustration. In: Proceedings of the 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). IEEE; 2016:148-156.
51. Petersen K, Gencel C. Worldviews, research methods, and their relationship to validity in empirical software engineering research. In: 2013 Joint Conference of the 23rd International Workshop on Software Measurement and the 8th International Conference on Software Process and Product Measurement IEEE; 2013:81-89.
52. Garousi V, Petersen K, Özkan B. Challenges and best practices in industry–academia collaborations in software engineering: a systematic literature review. *Inf Softw Technol*. 2016;79:106-127.
53. Barney S, Petersen K, Svahnberg M, Aurum A, Barney HT. Software quality trade-offs: a systematic map. *Inf Softw Technol*. 2012;54(7):651-662.
54. Petersen K, Wohlin C. Context in industrial software engineering research. In: Proceedings of the Third International Symposium on Empirical Software Engineering and Measurement, (ESEM 2009). IEEE Computer Society; 2009:401-404.
55. Briand L, Bianculli D, Nejati S, Pastore F, Sabetzadeh M. The case for context-driven software engineering research: generalizability is overrated. *IEEE Softw*. 2017;34(5):72-75.
56. Dybå T. Contextualizing empirical evidence. *IEEE Softw*. 2013;30(1):81-83.
57. Jadad AR, Haynes RB, Hunt D, Browman GP. The internet and evidence-based decision-making: a needed synergy for efficient knowledge management in health care. *Cmaj*. 2000;162(3):362-365.
58. Djulbegovic B, Guyatt GH. Progress in evidence-based medicine: a quarter century on. *Lancet*. 2017;390(10092):415-423.
59. Lemos AL, Daniel F, Benatallah B. Web service composition: a survey of techniques and tools. *ACM Comput Surv*. 2015;48(3):1-41.
60. Hasselbring W. Software architecture: past, present, future. *The Essence of Software Engineering*. Cham: Springer; 2018:169-184.
61. Woods E. Software architecture in a changing world. *IEEE Softw*. 2016;33(6):94-97.
62. LaToza TD, van der Hoek A. Crowdsourcing in software engineering: models, motivations, and challenges. *IEEE Softw*. 2016;33(1):74-80.
63. Hasselbring W, Steinacker G. Microservice architectures for scalability, agility and reliability in e-commerce. In: 2017 IEEE International Conference on Software Architecture Workshops (ICSAW) IEEE; 2017:243-246.
64. Glass RL, Vessey I. Contemporary application-domain taxonomies. *IEEE Softw*. 1995;12(4):63-76.
65. Blum BI. A taxonomy of software development methods. *Commun ACM*. 1994;37(11):82-94.
66. Unterkalmsteiner M, Feldt R, Gorschek T. A taxonomy for requirements engineering and software test alignment. *ACM Trans Softw Eng Methodol*. 2014;23(2):16:1-16:38.
67. Bourque P, Fairley RE, eds. *SWEBOK: Guide to the Software Engineering Body of Knowledge*. Version 3.0: IEEE Computer Society; 2014. http://www.swebok.org/
68. Kwasnik BH. The role of classification in knowledge representation and discovery. *Libr Trends*. 1999;48(1):22-47.
69. Usman M, Britto R, Börstler J, Mendes E. Taxonomies in software engineering: a systematic mapping study and a revised taxonomy development method. *Inf Softw Technol*. 2017;85:43-59.
70. Resnik MD. *Choices: An Introduction to Decision Theory*: University of Minnesota Press; 1987.
71. Simon HA. Rational choice and the structure of the environment. *Psychol Rev*. 1956;63(2):129-138.
72. Ayala C, Nguyen-Duc A, Franch X, Höst M, Conradi R, Cruzes D, Babar MA. System requirements—OSS components: matching and mismatch resolution practices—an empirical study. *Empir Softw Eng*. 2018;23(6):3073-3128.
73. Vetschera R. Preference-based decision support in software engineering. In: Biffl S, Aurum A, Boehm BW, Erdogmus H, Grünbacher P, eds. *Value-Based Software Engineering*: Springer; 2006:67-89.
74. ISO/IEC 9126. ISO/IEC Information technology—software product quality—part 1: quality model. Report: ISO/IEC FDIS 9126-1:2000; 2000.
75. ISO/IEC 25010. ISO/IEC "Software engineering—software product quality requirements and evaluation (SQUARE) quality model". report: ISO/IEC 25010:2011; 2011.
76. Adewumi A, Misra S, Omoregbe N, Crawford B, Soto R. A systematic literature review of open source software quality assessment models. *SpringerPlus*. 2016;5(1):19-36.

77. Al-Badareen AB, Selamat MH, Jabar MA, Din J, Turaev S. Software quality models: a comparative study. In: Proceedings of the International Conference on Software Engineering and Computer Systems. Springer; 2011:46-55.
78. Kitchenham BA. What's up with software metrics—a preliminary mapping study. *J Syst Softw*. 2010;83(1):37-51.
79. Kazman R, Klein M, Clements P. ATAM: Method for Architecture Evaluation, Software Engineering Institute (SEI), Carnegie-Mellon University, Pittsburgh, PA, USA; 2000.
80. Akatsu S, Fujita Y, Kato T, Tsuda K. Structured analysis of the evaluation process for adopting open-source software. *Procedia Comput Sci*. 2018;126:1578-1586.
81. Basili VR, Caldiera G, Rombach HD. Experience factory. *Encyclopedia of Software Engineering*: Wiley Online Library; 2002.
82. Jarke M, Pohl K. Establishing visions in context: towards a model of requirements processes. In: Proceedings of the International Conference on Information Systems. Association for Information Systems; 1993:23-34.
83. Ware C. *Information Visualization: Perception for Design*. 3rd ed. Amsterdam: Morgan Kaufmann; 2012.
84. Tang A, Liang P, van Vliet H. Software architecture documentation: the road ahead. In: Proceedings of the 9th Working IEEE/IFIP Conference on Software Architecture, WICSA 2011. IEEE Computer Society; 2011:252-255.
85. Zdun U, Capilla R, Tran H, Zimmermann O. Sustainable architectural design decisions. *IEEE Softw*. 2013;30(6):46-53.
86. Li Z, Liang P, Avgeriou P. Application of knowledge-based approaches in software architecture: a systematic mapping study. *Inf Softw Technol*. 2013;55(5):777-794.
87. Tang A, Avgeriou P, Jansen A, Capilla R, Babar MA. A comparative study of architecture knowledge management tools. *J Syst Softw*. 2010;83(3):352-370.
88. Ding W, Liang P, Tang A, Van Vliet H. Knowledge-based approaches in software documentation: a systematic literature review. *Inf Softw Technol*. 2014;56(6):545-567.
89. ISO/IEC 42010. ISO/IEC/IEEE "Systems and software engineering Architecture description". report ISO/IEC/IEEE 42010:2011; 2011.
90. Jansen A, Bosch J. Software architecture as a set of architectural design decisions. In: Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05) IEEE; 2005:109-120.
91. Kruchten P. What do software architects really do? *J Syst Softw*. 2008;81(12):2413-2416.
92. Abrahamsson P, Babar MA, Kruchten P. Agility and architecture: can they coexist? *IEEE Softw*. 2010;27(2):16-22.
93. Chesbrough HW. *Open Innovation: The New Imperative for Creating and Profiting from Technology*: Harvard Business Press; 2003.
94. Runeson P. Open collaborative data: using OSS principles to share data in SW engineering. In: Proceedings of the 41st International Conference on Software Engineering: New Ideas and Emerging Results, ICSE (NIER) 2019. IEEE / ACM; 2019:25-28.
95. Linåker J, Munir H, Wnuk K, Mols C-E. Motivating the contributions: an open innovation perspective on what to share as open source software. *J Syst Softw*. 2018;135:17-36.
96. Smite D, Moe NB, Sablis A, Wohlin C. Software teams and their knowledge networks in large-scale software development. *Inf Softw Technol*. 2017;86:71-86.
97. Wohlin C, Smite D, Moe NB. A general theory of software engineering: Balancing human, social and organizational capitals. *J Syst Softw*. 2015;109(11):229-242.
98. Budgen D, Brereton P, Drummond S, Williams N. Reporting systematic reviews: Some lessons from a tertiary study. *Inf Softw Technol*. 2018;95:62-74.
99. Budgen D, Kitchenham BA, Brereton P. The case for knowledge translation. In: Proceedings of the International Symposium on Empirical Software Engineering and Measurement (ESEM 2013) IEEE; 2013:263-266.
100. Le Goues C, Jaspan C, Ozkaya I, Shaw M, Stolee KT. Bridging the gap: from research to practical advice. *IEEE Softw*. 2018;35(5):50-57.
101. Badampudi D, Wohlin C, Gorschek T. Contextualizing research evidence through knowledge translation in software engineering. In: Proceedings of the Evaluation and Assessment on Software Engineering (ESEM 2019). ACM; 2019:306-311.
102. Cartaxo B, Pinto G, Vieira E, Soares S. Evidence briefings: towards a medium to transfer knowledge from systematic reviews to practitioners. In: Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement; 2016:57.
103. van Vliet H. *Software Engineering—Principles and Practice (3. ed.)*: Wiley; 2008.
104. Nurmuliani N, Zowghi D, Fowell S. Analysis of requirements volatility during software development life cycle. In: Proceedings of the 15th Australian Software Engineering Conference (ASWEC 2004), 13–16 April 2004, Melbourne, Australia. IEEE Computer Society; 2004:28-39.
105. Ristoski P, Paulheim H. Semantic web in data mining and knowledge discovery: A comprehensive survey. *J Web Semantics*. 2016;36:1-22. Web Semant Sci Serv Agents World Wide Web.
106. Kasoju A, Petersen K, Mäntylä M. Analyzing an automotive testing process with evidence-based software engineering. *Inf Softw Technol*. 2013;55(7):1237-1259.

How to cite this article: Wohlin C, Papatheocharous E, Carlson J, et al. Towards evidence-based decision-making for identification and usage of assets in composite software: A research roadmap. *J Softw Evol Proc*. 2021;33:e2345. https://doi.org/10.1002/smr.2345