Received: 24 October 2018 | Revised: 1 February 2019 | Accepted: 17 February 2019
DOI: 10.1002/smr.2169

SPECIAL ISSUE - EMPIRICAL PAPER

Continuous Software Engineering and its Support by Usage and Decision Knowledge: An Interview Study with Practitioners

Jan Ole Johanssen[1] | Anja Kleebaum[2] | Barbara Paech[2] | Bernd Bruegge[1]

[1]Department of Informatics, Technical University of Munich, Munich, Germany
[2]Institute of Computer Science, Heidelberg University, Heidelberg, Germany

Correspondence
Jan Ole Johanssen, Department of Informatics, Technical University of Munich, Munich, Germany. Email: jan.johanssen@tum.de

Summary

Continuous software engineering (CSE) emerged as a process that is increasingly applied by practitioners. However, different perceptions impede its adoption in industry. Furthermore, opportunities through utilizing usage and decision knowledge remain unexploited. We conducted a semi-structured interview study with 24 practitioners from 17 companies to study how practitioners apply CSE during software evolution and how usage and decision knowledge can support CSE. Regarding the application of CSE, we identified five perspectives on CSE with tool- and methodology-driven definitions most prevalent. Automated tests, involved users, and shared rulesets are perceived as most relevant for CSE. Practitioners report more positive than negative experiences; however, more than half of their responses were neutral. Practitioners' future plans for CSE focus on enhancement, expansion, and on-demand adaption. Regarding the integration of usage and decision knowledge into CSE, practitioners perceive accountability and traceability as major benefits, while raising concerns about its feasibility and user groups. As short-term extensions, practitioners expect improvements regarding automation and role aspects, while long-term additions to integration and experimentation capabilities are demanded. We conclude that CSE remains partially difficult to capture for practitioners, while their attitude toward integrating usage and decision knowledge into CSE is positive.

KEYWORDS:
continuous software engineering, interview study, agile software development, usage knowledge, decision knowledge, tool support

1 | INTRODUCTION

Continuous software engineering (CSE) bundles activities, such as continuous integration and delivery, to enable continuous learning and improvement by frequently iterating on software increments [1,2]. This classifies CSE as a software engineering process [3]. Krusche and Bruegge address the need for a formal description of the continuous aspects of CSE to enable its adoption in real world settings [4]. Similarly, researchers highlight challenges in the introduction and enhancement of CSE in companies [5,6]. Practitioners need a starting point in order to approach CSE, since they might lack insight into interrelationships, potential risks, and challenges [7,8]. Comparing their CSE process with what other companies have implemented allows practitioners to assess their own progress. Likewise, practitioners can benefit from guiding principles to establish CSE in their company [2]. In order to derive such guidance, it is necessary to review the state-of-the-practice of CSE as a process. Since there are no previous empirical studies with practitioners that address CSE as a process, we conducted 20 interviews with 24 practitioners from 17 companies between April and September 2017. We studied various aspects that are related to CSE, i. e., how practitioners understand CSE. In addition, we focus on how practitioners utilize usage and decision knowledge, since these are important knowledge types that developers require to successfully evolve a software to the users' satisfaction. CSE offers new opportunities for a better management and utilization of usage and decision knowledge. Our overall goal is the integration of usage and decision knowledge in CSE to support software evolution[9,10]. In this article, we present findings on how practitioners perceive CSE in the industry. In addition, we develop and discuss a vision that describes how CSE can be supported by usage and decision knowledge. We describe the practitioners' opinions regarding its benefits, obstacles, short-term extension, and long-term addition proposals.

The contribution of our work consists of five parts. First, we report on study data that provides insights into the characteristics of companies, practitioners, as well as projects in the context of CSE. Second, we describe a set of 19 observations that are derived from practitioners' responses and which detail practitioners' perspectives, most relevant elements, experiences, and future plans for CSE. Third, we introduce and discuss a model to describe the components of CSE—the *Eye of CSE*. The model contains CSE elements and categories, highlights relations among them, and aims to support practice. We compare our findings to those of related empirical studies. Fourth, we describe our vision of integrating usage and decision knowledge into CSE and summarize the practitioners' opinions regarding benefits, obstacles, short-term extensions, and long-term additions of the presented vision. Furthermore, based on their responses, we derive their overall attitude toward the vision. Fifth, we discuss an improved version of the initially presented vision which incorporates the practitioners' responses, in particular with respect to the feedback on extensions and additions.

This article is structured as follows. Section 2 describes how we derived CSE elements and categories and presents the initial version of our approach for integrating usage and decision knowledge into CSE. We introduce our research questions in Section 3 along with the applied research method and elaborate on threats to the validity of our study. Section 4 provides descriptive statistics regarding companies, practitioners, and projects. Major observations on how practitioners apply CSE are presented in Section 5. We discuss our observations in Section 6 by describing the *Eye of CSE* and providing an overview of similar studies in the context of CSE. Section 7 presents the results of practitioners' impressions about usage and decision knowledge, while Section 8 discusses the implications for an improved version of the approach. Section 9 concludes the paper.

This article is an extended version of work published by Johanssen et al.[11]. The extension addresses the integration of usage and decision knowledge in CSE by introducing RQ2, adaptions throughout the article, extensions in Section 2 and 3, and new results and their discussion in Sections 7 and 8.

2 | FOUNDATIONS

In this section, we provide the foundation for understanding the context of our research questions. Therefore, we divide CSE into elements and categories. Furthermore, we describe an approach for a systematic integration of usage and decision knowledge into CSE.

2.1 | Categories and Elements of Continuous Software Engineering

Bosch *et al.*[1,5] define the *Stairway to Heaven* covering four major steps that a company needs to take on the way from traditional software development toward CSE: adopting agile practices, continuous integration of work, continuous deployment of releases, and advancing toward research and development as an innovation system. Fitzgerald and Stol[2] provide a holistic view on activities from business, development, and operation. Similar to the last step of the *Stairway to Heaven*, they state that CSE is more than adopting continuous delivery and continuous deployment. For them, CSE goes beyond the *DevOps* approach that demands that software development and its operational deployment are tightly integrated. They coined the term *BizDevOps* as a synonym for CSE to emphasize the need to improve the link between business strategy and software development. Based on this work[1,5,2], we assembled characteristics typical for CSE. We refer to them as *CSE elements* separated by *CSE categories* as listed in Table 1.

TABLE 1 CSE categories and CSE elements derived from Bosch *et al.*[1,5] and Fitzgerald and Stol[2].

CSE Categories	CSE Elements
User	Involved users and other stakeholders; learning from usage data and feedback; proactive customers
Software Management	Agile practices; short development sprints; continuous integration of work; continuous delivery; continuous deployment of releases
Development	Continuous planning activities; continuous requirements engineering; focus on features; modularized architecture and design; fast realization of changes
Code	Version control; branching strategies; fast commit of code; code reviews; code coverage
Quality	Automated tests; regular builds; pull requests; audits; run-time adaption
Knowledge	Sharing knowledge; continuous learning; capturing decisions and rationale

The frequent involvement of users is a major concept in CSE. Thus, we introduced *user* as a CSE category that refers to both customers who commissioned a project and end-users. *Software management* includes practices concerning the overall software process. The *development* category is composed of more specific development activities, such as requirements engineering and design excluding implementation and quality assurance. The *code* category includes implementation-related practices, such as *version control* and *branching strategies*. We bundled activities such as *audits* and *pull requests* in the *quality* category. The *knowledge* category collects practices supporting the overall knowledge management.

We acknowledge that the allocation of elements to categories is not always straightforward. For instance, we consider arguably technical practices, such as continuous delivery, under software management, in order to highlight their impact on the overall CSE process. Similarly, we include code reviews in the code category to emphasize their operational character, while pull requests, i. e., merging code, are viewed as quality-related tasks. We refined the list of CSE elements and categories based on the interview observations and discuss the issue of ambiguities in Section 6.

2.2 | Toward the Integration of Usage and Decision Knowledge into Continuous Software Engineering

We see CSE as an opportunity to support stakeholders in capturing, exploiting, and understanding usage and decision knowledge [9]. Both knowledge types are essential for software evolution and can contribute to an improved product quality as described in the following.

Short feedback cycles in CSE enable close involvement of users since they have access to the most recent version of the software system. Thus, **usage knowledge** provides insights into the users' acceptance of a software increment. Usage knowledge occurs in different forms, such as *explicit* or *implicit* feedback, that is either *pushed to* or *pulled by* developers [12]. It can be a valuable source for the validation and derivation of requirements [13]. Usage knowledge helps to improve the usability of a software system, however, user-centered design techniques are "[...] underused, difficult to master, and [..] inaccessible to common developers and small and medium-sized software development teams" [14]. With its lightweight and agile character, CSE provides the opportunity to close this gap between software engineering and usability practices.

When implementing requirements or fixing bugs, developers solve issues and thereby make decisions. During CSE, developers implement these decisions while keeping the software system in a releasable state of a high quality [4]. In order to make decisions that contribute to the high quality of the software system, developers need **decision knowledge**. Decision knowledge covers knowledge about decisions, the issues they address, solution proposals, their rationale, and their context. If developers lack such decision knowledge, they are likely to contribute to the erosion of the software architecture or introduce other quality problems. Capturing decision knowledge has many benefits, e. g., it improves decision-making through making criteria explicit and prevents knowledge vaporization [15]. However, decision knowledge is often not captured in practice [16]. CSE offers new opportunities to overcome this capture problem since it provides multiple practices and documentation locations in which developers can capture decision knowledge [17,18]. For example, developers capture decisions when they commit code in commit messages [19].

Given the benefits of usage and decision knowledge for CSE, we developed a vision of how these knowledge types can be better integrated. As part of the research project CURES ("Continuous Usage- and Rationale-based Evolution Decision Support"), we develop an extension to a **CSE Infrastructure** which enables to systematically manage usage and decision knowledge during CSE [9]. In Figure 1, we outline the major components of the CURES vision and describe them in the following in more detail.

This infrastructure incorporates the CSE elements given in Section 2.1 and refined in Section 6.1 except for the elements of the user and knowledge categories. **Developer**s and **User**s are the two main stakeholders. An event-based environment enables the developers to create new releases of software increments and to deliver them to the users at any time [20,21]. Developers organize their work following a well-defined strategy: they use feature branches to develop a specific software increment and they can merge back a feature branch into a master branch as soon as they have completed a feature under development. A proposal $\mathtt{P}$ forms the basis to start work on a feature $\mathtt{F}$. For instance, the proposal $\mathtt{P}_1$ initiates work on the feature $\mathtt{F}_1$. After the first commit, the current state of the feature branch is delivered through a release $\mathtt{R}_1$ to users, who are able to use it in their target environment. A **Monitoring and Feedback** component collects information about the release in operation and offers the possibility to the user to give explicit feedback $\mathtt{FB}$. In the running example, $\mathtt{FB}_1$ is provided by the user. Given this feedback, the developer makes a decision $\mathtt{D}_1$ to merge back the feature into the master branch, and thereby closing their work on the feature. The core of the CURES vision is represented by the **Knowledge Repository**. The knowledge repository continuously stores and relates information collected from the CSE infrastructure as well as from the monitoring and feedback component. A **Dashboard** component enables to access the stored knowledge. Developers use the dashboard to interact and reflect with the knowledge in order to improve their decision-making and development process in the CSE infrastructure.

The CURES vision supports more complex scenarios. For example, two proposals—$\mathtt{P}_{2a}$ and $\mathtt{P}_{2b}$—can simultaneously lead to the development of the feature $\mathtt{F}_2$ in two different ways. Based on the retrieved feedback $\mathtt{FB}_2$ and $\mathtt{FB}_3$, the developer puts the development of $\mathtt{F}_{2a}$ on hold and makes the decision $\mathtt{D}_2$ which leads to the merge of $\mathtt{F}_2$ into the master branch. By utilizing the dashboard, the knowledge repository, as well as the monitoring and feedback component, the CURES vision extends the CSE infrastructure with support for usage and decision knowledge. This enables a *continuous knowledge activity* that should improve software evolution and the overall quality of software products.

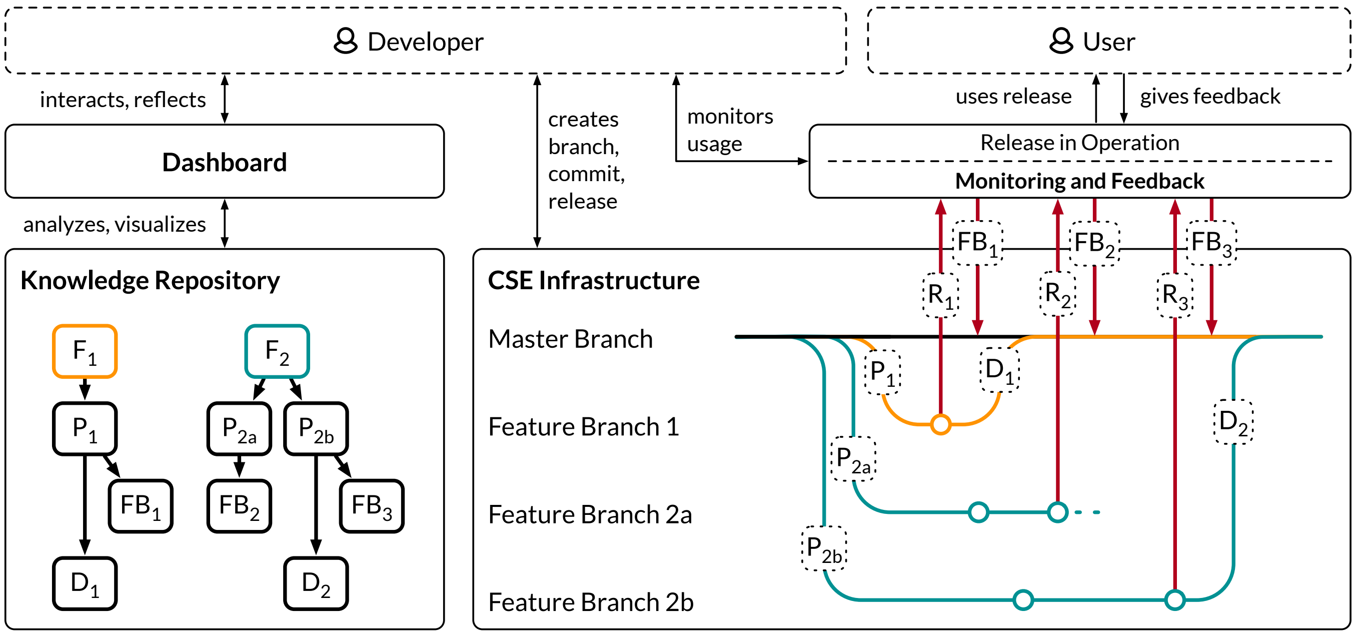


FIGURE 1 The initial CURES vision as introduced in our previous work[9]. The development of a feature `F` is initiated by a proposal `P`. Individual software increments can be delivered to users through a release `R`. While or after using a release, a user provides feedback `FB`. A decision `D` might lead to merging a feature into the master branch or capture other rationale regarding the feature under development.

3 | STUDY DESIGN

This section describes our research questions which we split into two areas of interest. Furthermore, we present our research method which we split into three phases. In addition, with respect to our study, we describe threats to its validity which were split into four aspects.

3.1 | Research Questions

Based on the foundations described in the preceding section, we study two areas of interest for this research.

RQ1 **How do practitioners apply CSE during software evolution?** — A major goal of this interview study is to understand how practitioners apply CSE during software evolution. From this goal, we derived the following four research questions that are based on the collection of CSE elements and categories introduced in Section 2.1.

RQ1.1 **How do practitioners define CSE?** — With this core question, we intend to learn about the practitioners' perception of CSE. Further, we want to know whether practitioners define a threshold that needs to be passed before a company can claim to practice CSE.

RQ1.2 **Which CSE elements are perceived as most relevant by practitioners?** — To understand the perception of CSE in more detail, we asked the practitioners about the three CSE elements most relevant to them. In addition, we collected applied tools.

RQ1.3 **What are practitioners' experiences with CSE?** — With this research question, we want to reveal positive, neutral, and negative experiences with the CSE elements. This is of particular interest for the practitioners that plan to adopt them as well.

RQ1.4 **What are practitioners' future plans for CSE?** — We asked for planned additions in the short and long term in order to understand trends of future CSE elements adoption.

RQ2 **How can usage and decision knowledge support practitioners during CSE?** — As described in Section 2.2, usage and decision knowledge are suitable candidates to be included in the CSE processes to improve the product increments. We were interested in the practitioners' thoughts of the CURES vision. To understand these thoughts, we derived the following five research questions.

RQ2.1 **What is the attitude of practitioners toward the CURES vision?** — Apart from detailed aspects in the following questions, we were interested in the practitioners' overall impression and their opinion on the feasibility of the CURES vision.

RQ2.2 **What are benefits of the CURES vision perceived by practitioners?** — This research question intends to find the core benefits of the integration of usage and decision knowledge into CSE. The practitioners' responses strengthen the goal of the CURES vision.

RQ2.3 **What are obstacles of the CURES vision perceived by practitioners?** — By asking the practitioners for major obstacles of the CURES vision, we detailed the feasibility aspects of the proposed approach. Responses help to understand practitioners' problems.

RQ2.4 **What are important extensions to the CURES vision according to practitioners?** — Based on their perceived benefits and obstacles, we asked the practitioners to provide ideas for short-term extensions to the CURES vision that do not require major changes.

RQ2.5 **What are potential additions to the CURES vision according to practitioners?** — Following the long-term visions of the practitioners, we strive for feature requests that could improve the CURES vision, while requiring significant changes to the overall idea. In comparison to RQ1.4, this research question has a focus on knowledge during CSE. Furthermore, it requires the practitioners to relate their answers to the CURES vision.

3.2 | Research Method

We performed a semi-structured interview study [22,23] and organized our study into the three phases *design and planning*, *data collection*, and *data analysis*. Each phase is described subsequently; the first two authors were equally involved in each of the phases.

3.2.1 | Design and Planning

We conducted a semi-structured interview study, since we focus on knowledge that resides in the minds of practitioners rather than in documents [23]. Furthermore, interviews allowed us to clarify problems right away and collect more information from the practitioners. We prepared a questionnaire containing descriptive data questions and interview questions derived from the research questions. The questionnaire contained multiple open questions that included sub-questions to further stimulate verbose answers by the practitioners. Figure 2 states example questions. We actively encouraged the interviewees to give detailed answers. We planned 90 minutes for the interview, which included research questions that are not further addressed in this article.

We assembled a list of companies who to our knowledge apply the majority of our preliminary collection of CSE elements (Table 1). We formulated a template request mail for scheduling an interview. We attached a slide deck sketching the CSE elements and categories. We asked interview partners to agree to the interview only, if they have worked in at least one project that applied the majority of the CSE elements. We did not restrict interview partners by role descriptions. However, we provided examples of roles that we preferably address, e. g., developer or project manager.

Since two authors conducted the interviews simultaneously, we set up an interview guideline to ensure comparability. Besides the interview questions, the guideline comprised remarks to increase the questions' understandability. We ran two dry runs with colleagues who have industry experience to practice the interview procedure.

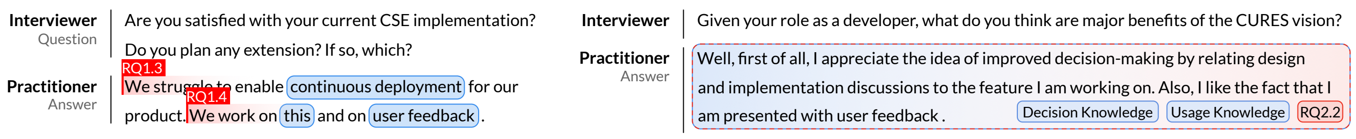
3.2.2 | Data Collection

We sent the interview requests to 22 companies, of which 18 replied. The first two authors conducted 19 interviews between April and June 2017 and one additional in September 2017. Half of the interviews were conducted in person; the others via phone. Descriptive data about the study participants are provided in Section 4. The interviews took 70 minutes on average and were audio-recorded with the permission of the interviewees. We transcribed the audio recordings and sent the transcripts to the interviewees to correct misunderstandings. During some interviews—especially with respect to RQ2—the interviewees gave us notably relevant feedback after the interview, for example on the way out or after some time via mail. We collected and added this information to the interview protocol to treat it like the rest of the interview, i. e., by systematically allocating answers to research questions and coding the answers, as described in the following Section 3.2.3. We guaranteed the anonymity of the practitioners by only publishing aggregated results.

3.2.3 | Data Analysis

Two authors of this paper analyzed the transcripts [24]. We utilized a *qualitative data analysis* software to apply two stages, as indicated with blue and red color in Figure 2. During the first stage, we allocated answers to a research question. Hereafter, we performed a fine-grained coding stage. The allocation to research questions was always made at sentence-level, the coding at word-level for RQ1 and sentence-level for RQ2.

For the first stage, i. e., allocating answers to research questions, both authors analyzed a single interview to measure the *intercoder reliability*; representativeness and completeness were criteria for choosing the interview. One author found 85 answers related to the research questions, the other 77. Given the total number of 162 instances, the authors matched in 134 and mismatched in 28, leaving a result of 82.72 % in equally allocated answers. The 28 mismatched instances were jointly discussed and resolved by mutual consent. The discussion necessary for this purpose


Interviewer Question: Are you satisfied with your current CSE implementation? Do you plan any extension? If so, which?

Practitioner Answer: RQ1.3 We struggle to enable continuous deployment for our product. RQ1.4 We work on this and on user feedback.

Interviewer: Given your role as a developer, what do you think are major benefits of the CURES vision?

Practitioner Answer: Well, first of all, I appreciate the idea of improved decision-making by relating design and implementation discussions to the feature I am working on. Also, I like the fact that I am presented with user feedback. Decision Knowledge Usage Knowledge RQ2.2

FIGURE 2 Two example interview extracts. The left side exemplifies practitioners' answers to RQ1 (red), while these in turn may contain multiple occurrences of fine-grained codes (blue). The right side shows an example for RQ2, in which one sentence could relate to multiple RQs and codes.

strengthened the shared understanding of the authors. We observed that almost all mismatches were caused by a missing allocation, not by the allocation to different research questions. In case of doubt, we agreed on allocating multiple research questions to an answer to prevent information loss. After the allocation of answers to the questions of RQ1, we updated our initially elected list of CSE elements in Table 1, based on the insights we derived from the answers. The updated collection is presented in Section 6.1, along with the rationale for the applied changes. After the allocation of answers to the questions of RQ2, we decided to update ten of these allocations due to a more sharpened understanding of minor differences, such as regarding short-term extensions (RQ2.4) and long-term additions (RQ2.5), which sometimes appeared to be similar and only differed in minor nuances, or were actual benefits (RQ2.2) overseen by the practitioners. The allocated answers to research questions form units for the second stage.

The second stage covered the coding of these answers. For RQ1, we used the updated collection of CSE elements as fine-grained codes. For RQ2, we coded the answers in terms of their focus; we distinguished between a *usage knowledge focus*, *decision knowledge focus*, or *other focus*. The coding was done manually, since searching for keywords is insufficient, given that practitioners use varying formulations to describe the same aspect. For instance, in Figure 2 on the left side, *this* refers to the CSE element *continuous deployment*, leaving the decision to add a code up to the author. Occasionally, the authors had to decide which code to use based on the context. We practiced the coding and agreed to prefer to code an instance if in doubt. Each author coded their own interviews. We analyzed the results quantitatively (Figure 4, 5, and 7). In case we coded a practitioner's answer to a research question more than once with the same code, we recorded only a single occurrence to receive binary results. We analyzed the interviews on the interview-level, and not on person-level, which means that—in case two interviewees participated in an interviewee—their answers were treated as one subject. Further, we analyzed the results qualitatively. With respect to RQ1, we summarized the results to describe observations (Section 5). With respect to RQ2, we summarized the results to form topics (Section 7). At least answers from two individual interviews were required to form an observation or a topic, respectively. Answers from a single interview can account for more than one response per topic, while one response can relate to more than one knowledge focus—which is relevant for Tables 2, 3, 4, and 5. An initial version of this paper was sent to the interviewees to validate the interview results.

3.3 | Threats to Validity

We discuss the study's threats to validity according to the four aspects of validity by Runeson *et al.*[23]. In case the collection and assessment of data for a specific research questions posed an individual threat, we provide more details on it in the related section, such as in the case of Section 5.3.

Construct validity concerns the disparity between the intended and the actual study observations[23]. First, the practitioners resemble a heterogeneous group, each having its own point of view on CSE. The conformance between them might be small. We tried to address heterogeneity by describing observations instead of facts. Second, the questions may be interpreted by practitioners in a different way than intended by us. We tried to minimize this possibility by conducting two interviews with colleagues. We discussed these interviews afterwards to reveal potential misinterpretations. Also, the format of the interviews allowed practitioners to ask questions at any time. Third, the authors might have influenced the participants by asking specific questions. To mitigate this risk, we used open-ended questions to elicit as much information as possible from practitioners. Finally, the collection of CSE elements is based on a model that is an abstraction of reality and—to some extent—subjective. We asked practitioners to describe their experiences with the proposed set of CSE elements, which might have biased them. We tried to mitigate this risk by collecting additional CSE elements.

Internal validity concerns correlations between the investigated factors and other factors[23]. The practitioners might have provided answers that do not fully reflect their daily work, since they were aware that the results would be published. We addressed this possibility by guaranteeing the full anonymity of interviewees and companies. Further, the interpretation of answers might be biased by the authors' *a priori* expectations and subconscious impressions. We addressed this threat by coding the transcriptions and discussing the codes. Finally, the slides might have biased the practitioners' perception of CSE. We perceive this as a minor threat, since it only affects RQ1.1 and they might have prepared beforehand anyway.

External validity addresses the generalizability of the study results[23]. We contacted companies that we already knew, which affects the sampling and might result in a selection bias. However, there is no central register of companies that apply CSE[25]. Clearly, this amounts to a risk to the representativeness of the participants. It is mitigated by the fact that the authors are from two different universities. Further, the diversity of projects and participants reinforces the generalizability. Finally, interviews are subjective, since they rely on the practitioners' statements. To reduce subjectivity, we conducted 20 interviews, to acquire a wider set of opinions.

Reliability validity concerns the study's dependency on specific researchers[23]. After we carried out coding training and checked intercoder reliability, two authors individually coded different transcripts. We address this threat by discussing questions during coding. In addition, a third author of this paper supervised the interview analysis.

4 | DESCRIPTIVE STUDY DATA

In this section, we report on descriptive data about the companies, practitioners, and projects that were analyzed. Figure 3 visualizes a summary of the following subsections. Overall, we interviewed 24 practitioners from 17 companies during 20 interviews. One company was interviewed twice, another one three times. We aimed for a diverse composition of interviews through companies varying in size, practitioners of different roles, and projects from various domains.

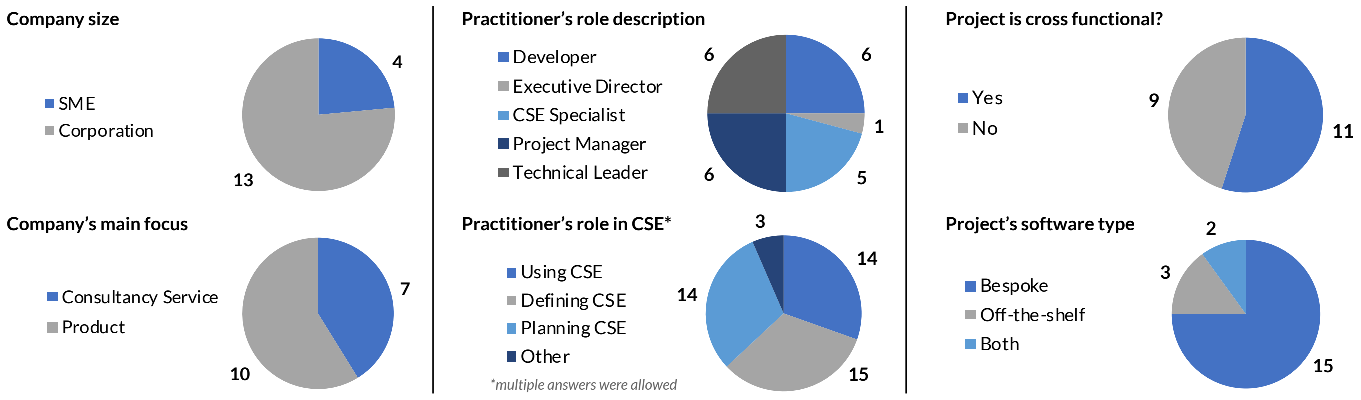


FIGURE 3 Descriptive data of the interview study: 24 practitioners from 17 companies were interviewed on 20 projects. In each interview, the practitioners related their answers to one particular project; four interviews were attended by two practitioners.

4.1 | Companies

Four companies (24 %) are considered as small and medium-sized enterprises[1] (SME), which means a maximum staff headcount of 250. We call the remaining 13 companies corporations, while they could be further categorized in companies of up to 2.000 (8), around 50.000 (2), and 100.000 or more employees (3). We report the overall number of employees, since we assume that the CSE process is not limited to a specific role from the development team. Seven (41 %) of the interviewed companies offer consultancy services, mostly to other businesses, while ten (59 %) companies develop software products for the consumer and business markets.

4.2 | Practitioners

Based on their role description, we grouped the 24 practitioners into five categories: *CSE specialists* (21 %), a role with a CSE reference, e. g., a continuous deployment manager or a DevOps engineer, *developers* (25 %), *project managers* (25 %), a role with project-focused responsibilities, and *technical leaders* (25 %), a role with technical-focused responsibilities; one practitioner reported as an *executive director*. On average, the practitioners have spent two years in the respective role. All practitioners hold a Bachelor or Master degree with three-quarters in a field in or close to computer science. With one lacking response, on average, 23 practitioners have an experience in IT projects of 10 years and participated in 19 IT projects.

We asked the practitioners whether they see themselves in one of the following three roles: *using*, *defining*, or *planning* CSE. Practitioners in a *using role* frequently apply and benefit from CSE, e. g., developers who regularly commit code. Practitioners in a *defining role* set rules on how CSE elements are applied, e. g., whether a code integration is triggered by an event, such as a commit, or on an hourly basis. It is the responsibility of a *planning*

[1]http://ec.europa.eu/growth/smes/business-friendly-environment/sme-definition

role to think ahead and take future addition to a CSE environment into consideration. Many practitioners saw themselves in multiple roles: 14 using, 15 defining, and 14 planning. Seven practitioners reported to adhere to all three roles—*I am everything* or *straight through*. Six other practitioners grouped themselves into two roles at the same time: In one role, they collect knowledge regarding a CSE element and in the other role they share it with other practitioners. Six practitioners adhered to a single role only. Two developers saw themselves solely in a using role; a project manager and a CSE specialist classified themselves in a defining and planning role, respectively. Two other practitioners did not see themselves in one of the three roles provided. They and a third practitioner proposed an additional role: *promoting*. This role pushes CSE efforts forward, in particular in situations in which it does not appear reasonable from a time or cost perspective—but would pay off in the long run.

4.3 | Projects

For each interview, we asked the practitioners to select one project that they are currently working on or which contains several CSE elements. On average, 20.25 employees work in a project, for SMEs 10.0 and for corporations 22.81. Eleven practitioners (64.71 %, including all SMEs) consider their project as cross functional, e. g., involving several other stakeholders from within the company, such as marketing professionals to represent the users. Note that practitioners from all four SMEs stated a cross functional project structure. Three-quarters (15) of the projects develop *bespoke* software, e. g., custom software. Three projects (all by SMEs) develop commercial off-the-shelf software; two projects target both types. We asked the practitioners, if their projects and the way they have to be approached need to comply with any legal, security, medical, or environmental factors; every second practitioner confirmed this. However, some practitioners referred to the rules for the product, rather than the project.

5 | RESULTS ON HOW COMPANIES APPLY CSE (RQ1)

In this section, we present the results on RQ1 introduced in Section 3.1. We illustrate the results of our quantitative analysis of CSE elements and categories mentioned in the interviews in Figures 4 and 5. They relate to the model in Section 6.1. The following subsections address the research questions based on both figures: At the beginning of each subsection, we answer a research question and then provide a more detailed analysis by stating several observations.

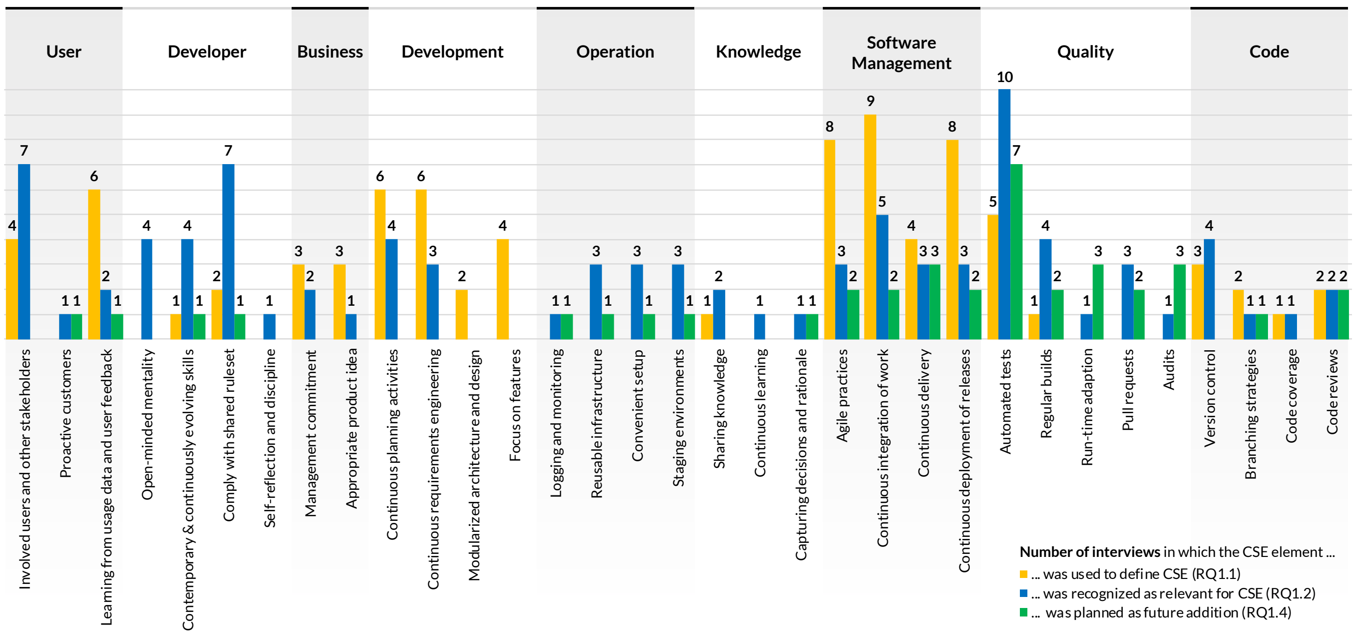


FIGURE 4 How often during an interview is a CSE element mentioned regarding a particular research question? Yellow bars indicate the number of interviews in which the respective CSE element was used for defining CSE (RQ1.1). Blue bars indicate the practitioners' tendency toward relevant elements for CSE (RQ1.2). Green bars indicate CSE elements intended as future additions (RQ1.4). Answers were summarized as one interview, in case multiple practitioners participated at the same time. The CSE elements are headed by their categories.

5.1 | Practitioners' Definition of CSE

Since the term CSE only emerged in recent years, we asked the practitioners how they define CSE. We plot the results in Figure 4.

RQ1.1 **How do practitioners define CSE?** We found that the practitioners definitions of CSE are mainly driven by CSE elements from the software management category, i. e., continuous integration of work, agile practices, and continuous deployment of releases. CSE elements from the development and user category were also mentioned repeatedly. Based on the responses, we identified five perspectives that influence the definition of CSE: a *tool* (5.1.1), *methodology* (5.1.2), *developer* (5.1.3), *life cycle* (5.1.4), and *product management* (5.1.5) perspective.

Out of the 24 practitioners interviewed, slightly more than half (54 %) were using the term *CSE* as part of their active vocabulary. About two-thirds (66 %) of all interviewees gave a definition of their understanding of CSE. Notably, 75 % of the interviewees in SMEs both gave a definition and actively used the term *CSE*. For some practitioners, CSE is still ambiguous. They describe it as *fuzzy*, *abstract*, and *lacking a distinction*.

5.1.1 | Tool Perspective

Practitioners, i. e., six developers and CSE specialists, make use of tool descriptions when defining CSE. Their descriptions are built on statements such as *in that regard, company A provides tool B*, or *after introducing tool C, we were able to accomplish element D*, or *we are currently looking into tool E of company F*. Four practitioners explicitly highlight that it is a well-chosen *tool chain* that enables CSE. In their opinion, the successful accomplishment of the steps availability, integration, and usage of tools allow a company to claim to be implementing CSE.

Observation 1 *In particular developers and CSE specialists rely on a tool-driven approach to define CSE. Commercial tools influence their understanding.*

5.1.2 | Methodology Perspective

In more than half of the interviews, practitioners cite a methodological perspective to define CSE. They emphasize a focus on short iterations and feedback. Not mentioning specific tools, many practitioners highlight the importance of *how* the tools are applied. For instance, a sophisticated branching strategy should be preferred, instead of the exhaustive use of capabilities a version control system might offer. A state of *on-going iteration* should be reached, in which each commit leads to a finalized product. Different elements, such as continuous integration or agile practices, are applied to achieve a high level of automatization. Notably, some practitioners reflect on the combination of multiple CSE elements to achieve synergy effects. This implies that their perspective takes the impacts of CSE on other areas, such as the management of requirements, into account.

Observation 2 *Practitioners define CSE from a methodological perspective that aims for short iterations during software evolution. This perspective relies on well-defined steps in tool usage, workflows, and procedures. Every step enables a seamless workflow from a commit until its finalization in form of a build.*

Several practitioners mention the importance of instant feature visibility to users. This enables constant retrieval of user feedback on the latest releases and more iterations with input from outside are performed. One practitioner advocates that every CSE process should be designed in accordance with the goal of matching customers' requirements through the implementation of short feedback loops. Another practitioner advises the collection of feedback *as often as necessary, rather than as often as possible*. Observations 2 and 3 are related and depend on each other to reach their full potential. However, we observed that not all practitioners implement both, leaving opportunities for improvement.

Observation 3 *CSE makes changes instantly visible to users. As a result, user feedback can be elicited and used to match the software to the requirements.*

5.1.3 | Developer Perspective

Several developers, project managers, and CSE specialists suggest a developer-driven perspective on CSE. Similar to Section 5.1.2, they do not base their descriptions on specific tools. First, they emphasize that CSE enables developers to fully focus on their main task, i. e., developing software, rather than deal with other processes, such as infrastructure management. This includes increasing the speed of the development process by removing idle times. Second, CSE allows practitioners to better estimate and classify their daily tasks. It ensures that newly introduced changes do not break code—an aspect which is not only in the interest of the overall product, but also a factor in the mind of developers. Providing a safe environment to develop and test software is a major characteristic of CSE. Third, according to practitioners, CSE supports the detachment of recurring tasks from an individual; in particular, by introducing defined processes, knowledge vaporization can be prevented. Remarkably, one practitioner highlights the increased responsibility of developers when giving their definition of CSE. This relates to the fact that developers independently create and deploy releases which—to unlock the full potential of CSE—should take place without any clearance or dedicated release plan.

Observation 4 *CSE allows developers to focus on their tasks and creates a safe development environment. Specific tasks are removed from individuals.*

5.1.4 | Life Cycle Perspective

Various practitioners agreed that CSE opens up a new perspective on the software life cycle: development, deployment, and operational phases blend into each other. Practitioners report shorter intervals between the development and the production phases. They further state that CSE is characterized by the fact that a system's functionality is extended continuously and shaped by *continuous application life cycle management*.

Observation 5 *Practitioners characterize CSE by the blending of different phases of software engineering, such as development, deployment, and operation. According to their perception, this makes long-living systems easier to maintain.*

5.1.5 | Product Management Perspective

Project managers, technical leaders, and executive directors formulate a definition of CSE from a product perspective. In their opinion, CSE is represented by constant funding, provided to continuously improve a product. This includes the project managers' ability to continuously acquire new requirements as well as to create and re-prioritize product tasks. One technical leader admits that not every product is guaranteed to follow this pattern. According to this practitioner, the application of CSE cannot simply be defined for a project: it is the product that determines whether CSE can be applied or not. Most of the projects are designed to follow other software evolution practices, and not CSE in particular. A product's compatibility with CSE processes needs to be ensured before applying CSE. Further, the environment in which the user receives the product plays an important role, as pointed out by a practitioner from a large corporation. It is required to keep pace with the CSE practices as defined in observation 2. For instance, if the deployment of a product requires certain manual steps, the product itself cannot be developed using CSE processes. One practitioner defines CSE as the integration of customer, business model, software, and hardware. If a company successfully combines these four aspects, it is implicitly implementing CSE practices such as continuous integration and continuous delivery.

Observation 6 *Practitioners' definition of CSE is influenced by the product under development. Product-related factors such as funding, functionality, business model, and its future target environment need to match the continuous development capability.*

5.2 | Practitioners' Relevant Elements of CSE

We asked practitioners' for CSE elements that *drive* CSE. Thus, they were required to list the three—in their opinion—most relevant CSE elements. In case a practitioner mentioned a CSE element that was not part of Table 1, we recorded it as a relevant element. We plot the results in Figure 4.

RQ1.2 **Which CSE elements are perceived as most relevant by practitioners?** Practitioners perceive CSE elements from three categories as most relevant: *quality*, i. e., automated tests, *user*, i. e., involved users and other stakeholders, *developer*, i. e., compliance with a shared ruleset. Practitioners mention more CSE elements: in particular the developers consider elements from the *code* category, such as version control, as obligatory, pivotal, and indispensable to any further steps in CSE. This strengthens the first stair in the *Stairway to Heaven* model of Bosch *et al.*[1]. We summarize the results as follows: *user commitment* (5.2.1), *team commitment* (5.2.2), and *automated loop* (5.2.3).

5.2.1 | User Commitment

In particular practitioners from SMEs that develop off-the-shelf software highlight the contact with users as a CSE element. One technical leader points out a significant difference in the user audience: While there is a large number of users that are *passively* using software, i. e., users that do not state an interest in new additions that do not affect their typical workflows, it is the less represented *active* users who help to make the CSE feedback loop efficient and functional. One project manager continues this thought by stating that the interaction with the users is barely technically defined in CSE. They approach this issue by actively trying to involve users through the promotion of nightly and beta builds. Two technical leaders claim that the success of a CSE project depends to a great extent on the degree of user involvement—*if there is no involved user, you lose*. Some practitioners remark that *users do not know what they want until they see it*. Enabled by continuous delivery, users can frequently provide feedback and thereby steer the development process toward their needs.

Observation 7 *Practitioners perceive the users' commitment toward taking an active part in the development process as a relevant aspect of CSE.*

5.2.2 | Team Commitment

Many practitioners perceive the team and its commitment as highly relevant for CSE. According to one developer, it is important that team members are open-minded toward the development process and take an active role in its formation. They need to adhere to a shared ruleset to work successfully. This poses challenges, as noted in Section 5.3. Practitioners illustrate a need for the full support of managers and executives. They should provide their attention to the project and trust in the performance and skills of the team. Rather than tools, it is the methods and processes introduced by agile practices that bind team members together. A *continuous process improvement* activity should be carried out by the team.

Observation 8 *Practitioners perceive an open-minded team mentality that complies with a shared set of rules as the basis of successful CSE teams. Management commitment is indispensable, while agile practices serve as the main unifying factor.*

5.2.3 | Automated Loop

Half of the practitioners declare the automatization of process loops to be the core of CSE. According to these individuals, discrete phases should be replaced by short, compact loops. They use the term *continuous pipeline* to describe a well-defined, highly automated process, which can be further adapted to the characteristics of the product under development. The practitioners share a vision of a non-linear process that can either be serialized or else run in parallel. *Automated tests* is the most relevant CSE element for ten practitioners. Others mention continuous integration and continuous deployment as the major building blocks of an efficient automated loop comprising different fulfillment levels. Operational aspects, such as *staging environments*, complete their idea of an automated loop.

Observation 9 *Practitioners perceive a high maturity level of automatization as an essential aspect of CSE. This is enabled by well-defined steps that form a non-linear process model. Furthermore, practitioners state automated tests as the most relevant CSE element.*

5.3 | Practitioners' Experience with CSE

We asked practitioners about positive, neutral, and negative experiences with CSE elements. Figure 5 shows the results grouped by their respective categories. Note that not every practitioner provided an experience report and—given that we grouped the responses by the CSE categories—practitioners may be represented multiple times if they responded to more than one CSE element of the same category. Furthermore, we coded responses as either positive or negative only in cases in which clear indicators by the practitioners were provided, for instance if the practitioners used phrases such as *"we had problems with implementing <CSE element>"* or *"we could not work without <CSE element>"*. At the same time, the high number of neutral experiences could be understood as a questions-specific threat that is caused by the design of the question.

RQ1.3 **What are practitioners' experiences with CSE?** 19 positive, 56 neutral, 17 negative experiences with CSE elements were reported. Notably, more than 50 % of the positive experiences are stated by SMEs, while forming roughly a quarter of the interviewee sample. Categories with many positive experiences as in *code* and *software management* are an indicator for CSE elements that can serve as an entry point to CSE, since they may be easy to implement. Few positive mentions as is the case with *knowledge*, *business*, and *user* may be a sign of the low maturity of CSE elements. Neutral responses may indicate that practitioners are currently evaluating various CSE elements in the field. A large number of negative experiences as with the *developer* category indicates challenging CSE elements. We discuss distinct experience reports derived from five CSE categories: *developer* (5.3.1), *operation* (5.3.2), *software management* (5.3.3), *user* (5.3.4), and *quality* (5.3.5).

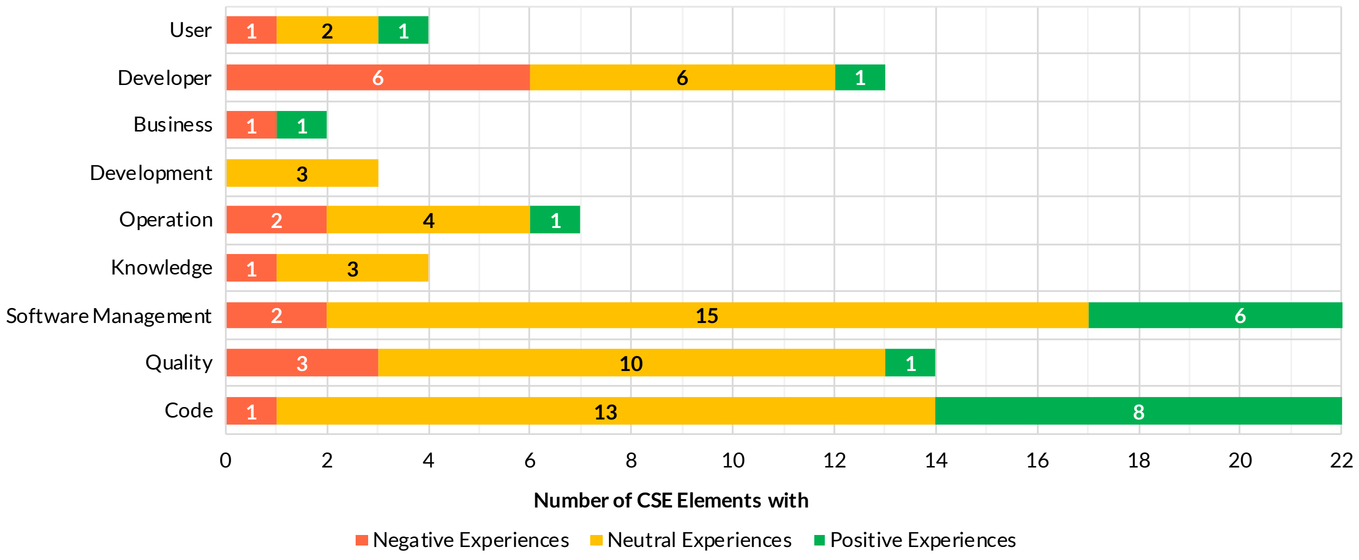


FIGURE 5 For RQ1.3, the practitioners' reported on negative, neutral, and positive experiences with CSE elements. The bar length indicates the number of CSE elements mentioned by practitioners. We grouped responses for individual CSE elements by their corresponding category.

5.3.1 | Developer

Most negative experiences were reported in the developer category, in particular for *complying with shared ruleset*s and *contemporary and continuously evolving skills*. Negative experiences are amplified by problems with other CSE elements, such as *branching strategies*. One practitioner reports problems when dealing with too many branches, which they consider *poisonous to continuous integration*. However, they admit that it is sometimes inevitable to have several branches, even though this situation can be approached with well-defined rules; for example, keeping the lifespan of a branch as short as possible, and committing code frequently. According to six practitioners, this demands attention from developers. Switching to a new way of developing software requires the willingness to evolve skills and extensive knowledge—which is why one practitioner views young graduates as having advantages over long-serving employees. One practitioner sketches solutions on how to overcome obstacles: providing incentives for successful work, enabling and supporting in-house training, as well as creating showcase projects. The practitioners agree that an open-minded mentality on the part of developers, as well as their ability to adapt and withstand the speed and frequency of CSE amount to both the basic requirement and also a major challenge for developers.

Observation 10 *Practitioners acknowledge a major challenge in the developers' capability to comply with shared rulesets and in their open-minded mentality to continuously evolve their skills.*

Two practitioners note that automatization makes developers use methods which they would otherwise dispense with. However, they admit that this makes it easier for developers to neglect other responsibilities; therefore, they stress that CSE demands *self-reflection and discipline*. Practitioners with a leading role state that they trust their team members. They initiate discussions to find a consensus whenever necessary.

Observation 11 *CSE does not solely build on the developers' skills, but also on their ability to reflect on their work and on their sense of responsibility.*

5.3.2 | Operation

Corporations struggle with legacy burdens, e. g., existing tool contracts that are not intended for CSE. Similarly, tools intended for CSE are used for other purposes, such as issue tracking systems for internal incident management, rather than for software development. Practitioners of other corporations emphasize the fact that tools are not a hurdle for them, since they can easily be bought—it is the integration that poses the challenges. Other corporations have to adhere to formal regulations that impede or prevent CSE from being applied in a given project, e. g., by relying on paperwork-driven processes. One practitioner states that there is a risk that agile projects could fall back into their previous static patterns.

Observation 12 *While practitioners are willing to apply CSE, it is their company's current set of tools that keeps them from making a complete transition and fully adapting CSE. Furthermore, requirements in regulated domains hinder the implementation of CSE.*

One practitioner complains that a major cost factor lies in setting up the infrastructure for new projects to use CSE elements. Furthermore, given the internal hierarchical and management structure, some corporations do not have the capacity to respond rapidly to changes within the project.

Observation 13 *Practitioners state that the successful implementation of CSE requires the ability to set up projects without major cost or time penalties.*

5.3.3 | Software Management

Practitioners report positive experiences with the implementation of *agile practices*. Building-blocks such as sprints, review meetings, or SCRUM-Boards are well-received and provide high value. Some difficulties arise during task prioritization, since only limited resources—time and money—are available. Apart from that, CSE elements, such as continuous integration, are essential to practitioners. One practitioner mentions significant synergy effects when using tools of the same vendor for issue tracking, source code management, as well as continuous integration and delivery.

Observation 14 *Practitioners attest that CSE elements related to software management, such as agile practices or continuous integration of work, are widely and successfully adopted in their projects.*

5.3.4 | User

CSE elements related to users are barely referenced whenever practitioners are asked about their experiences. The user's role is rated as *fuzzy* and there is a threat that user feedback does not continuously flow back to developers. One practitioner ascribes this to the fact that CSE does not produce major releases that are perceived as notable changes by the users. Thereby, user feedback is submitted late in the process in the form of incidents or change requests. Developers then lack traceability links to changes that caused the feedback. One project manager is concerned that software quality suffers from the release frequency in the short run since immature releases impede the users' confidence in the product.

Observation 15 *Practitioners have not yet created processes that interact with the users in a way similar to well-established practices such as continuous integration. This is mainly due to the fact that the users' responses to ongoing changes are difficult to record, trace, and assess.*

5.3.5 | Quality

Practitioners welcome CSE elements such as *pull requests* combined with *code reviews*. *Code coverage* and *audits* are practices that are gaining in importance. However, some practitioners raise concerns because quality metrics are not being tracked. We observed that practitioners' responses regarding the *quality* category are driven by various testing and exploration reports. First, every project strives for high software quality and therefore tries to invest effort into improving. Second, as there is no final release, processes to improve software quality can always be developed further. Third, the influence of changes to software quality might become apparent only at a later time.

Observation 16 *Practitioners have had varying experiences with quality elements during CSE, but they still invest into improvements.*

5.4 | Practitioners' Future Plans for CSE

We asked practitioners which CSE elements they plan to add in the future to discover future trends in CSE. We plot the results in Figure 4.

RQ1.4 **What are practitioners' future plans for CSE?** Practitioners' plans are vague and mostly distributed across elements. 19 CSE elements either received only one, two, or three mentions by the practitioners in the interviews. One CSE element stood out with seven mentions: *automated tests*. We found that the majority of practitioners described plans that span multiple CSE categories. We identified three main strategies in the practitioners' answers: *enhancement* (5.4.1), *expansion* (5.4.2), and *on-demand adaption* (5.4.3).

5.4.1 | Enhancement Strategy

Practitioners base their strategy for the future on a combination of the *methodology perspective* (observation 2) and *quality*, one of the most relevant categories mentioned (observation 9), yet one with mostly neutral experiences (observation 16). Seven practitioners mention automatization in the context of quality as one of their major plans for the short and long term. While *automated tests* are applied for some parts of the products, they should be made available for all. Two practitioners mention their plans of combining elements from the *operation* category, such as deployment in containers to enhance automatization. Three practitioners list activities to enhance their current state: code quality workshops, giving code metrics a meaning by calling for action rather than representing read-only information, and connecting CSE elements from different CSE categories.

One technical leader plans to bring the interaction with the user to the next level by detaching feedback collection from the individual—which is currently often the case—and creating a well-defined, high maturity level process similar to the one used for continuous integration. Other practitioners mention various ways of optimizing the implementation of agile practices or the application of branching strategies.

Observation 17 *Practitioners aim for a fully automated loop to increase software quality by applying goal-driven enhancements to existing CSE elements.*

5.4.2 | Expansion Strategy

The future plans of four practitioners can be summarized as an *expansion strategy*, i.e., applying recently established CSE elements to other areas of a project. The expansion of continuous delivery to more platforms is mentioned several times. This means adapting similar practices such as automatic deployment in mobile environments to their server-side counterparts. Similarly, expanding continuous integration to more platforms is mentioned several times; for example, one practitioner praises progress in *Java* environments, while they struggle with *JavaScript*. Another practitioner mentions the expansion of documentation to more areas than it is the case at the present time.

Observation 18 *Practitioners aim to extend efficient CSE elements to other areas of the project or similar products.*

5.4.3 | On-Demand Adaption Strategy

Three practitioners indicate a general interest in future CSE additions, however, they rely on an event-triggered or *on-demand* strategy to adapt, i.e., enhance or extend, their CSE elements. One practitioner describes an exploratory process in which CSE elements are added *step by step*. If they encounter a situation that would benefit from improvements, they initiate further investigations into possible solutions. Another practitioner makes the addition of further CSE elements dependent on the team's dynamic. A project manager highlights the effort in time that is required to implement certain CSE elements, making an overall process transition a time-consuming undertaking.

Observation 19 *Practitioners make enhancements and additions to CSE dependent on events that call for action. They postpone decisions for further additions to a later point in time.*

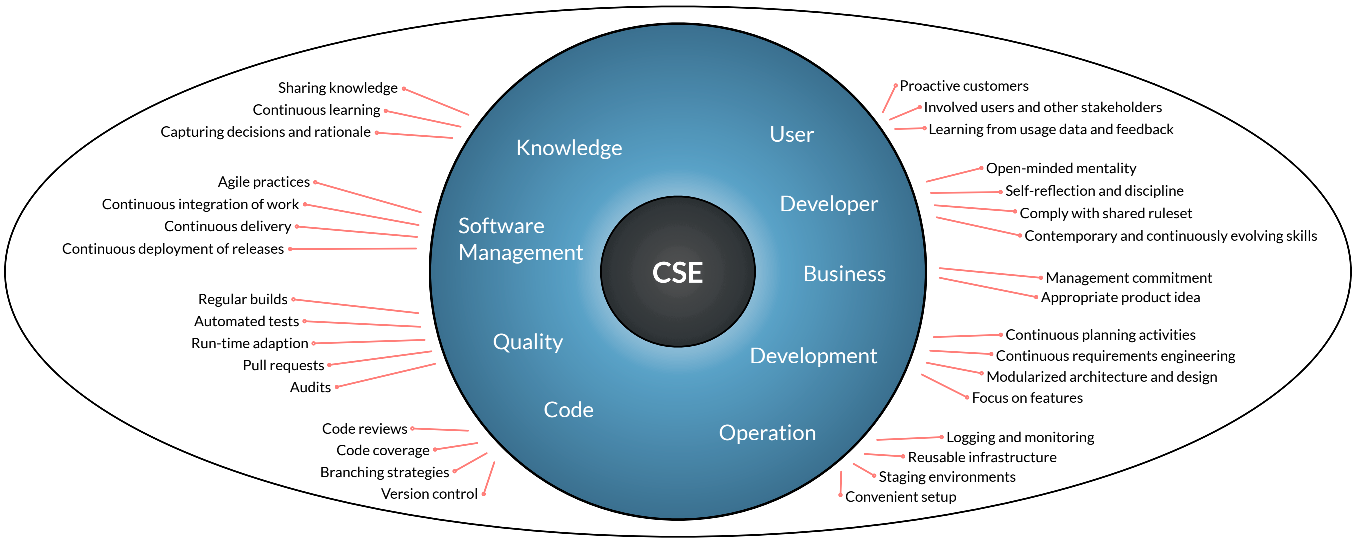


FIGURE 6 The model *Eye of CSE* consists of nine CSE categories and 33 CSE elements. The proximity of elements and categories suggests relationships between them. The model can open up one's eyes to new ideas for additions to current CSE processes.

6 | DISCUSSION ON HOW COMPANIES APPLY CSE (RQ1)

In this section, we summarize our insights in the form of a model and present related work. Above all, we find it necessary to state that some observations from this study appear obvious, ambiguous, or may even be seen as a tautology. However, since we are searching for empirical evidence on which to base assumptions that were previously only anecdotal, to some extent this leaves us with reporting *obvious conclusions*[25].

6.1 | Eye of CSE

Based on the practitioners' answers, we updated the CSE elements and categories in Table 1 by adding three new categories—*developer*, *business*, and *operation*—and removing two CSE elements, which are represented in the new categories. We refer to the final set of CSE elements and categories as the *Eye of CSE* and present a graphical representation in Figure 6. A well-established infrastructure as demanded by our CURES vision (Section 2.2) incorporates all of these CSE elements.

The eye's focus lies on a comprehensive implementation of CSE, represented by the pupil. The process of reaching this goal is influenced by the categories that are part of the eye's iris. The categories define the diameter and size of the pupil and thereby the perception of CSE. We learned from the interviews that CSE categories are intertwined and have fuzzy boundaries. This is partly different from the sequential nature of the *Stairway to Heaven* by Bosch *et al.*[1,5]: even if some CSE elements, such as continuous integration and delivery, require a step-wise introduction, the practitioners' statements suggest that CSE should be approached from multiple angles simultaneously. From the results of our interview study, we cannot advise a clearly defined sequence of adding CSE elements toward the implementation of CSE in a company. Instead, the *Eye of CSE* can serve as a checklist for practitioners to tackle the subject of CSE by incrementally applying CSE elements and keeping an eye on potential next steps.

During the design of the *Eye of CSE*, we strived to accurately allocate CSE elements to categories by analyzing the practitioners' answers and carrying out internal discussions. By connecting CSE elements to the iris and not directly to CSE categories, we acknowledge that one CSE element can relate to one or more categories—their allocation is rather loose. Therefore, the proximity of CSE elements within the *sclera*, the white of the eye, only suggests a relation to a category and among multiple CSE elements. The grouping of CSE categories allows practitioners to recognize relationships based on their proximity within the eye and the position of the CSE elements. In particular, when improving one category, it can be worth considering the addition of another category that features similar CSE elements.

Example 1 *Whenever practitioners expand on software management, the categories knowledge and quality should be incorporated, since their containing CSE elements are interrelated and have a positive effect on each other.*

The addition of one CSE element from a category can have a positive effect on one or more CSE elements from the same or another category. The joint addition of CSE elements could allow companies to benefit from synergies of the CSE elements' effect or to reduce the implementation efforts. However, this relationship cannot be derived from the model at its current state.

Example 2 *Audits can include code reviews, which makes them part of the quality category. Likewise, learning from usage data and feedback requires the developers' open-minded mentality. Addressing sharing knowledge is an example for a CSE element that has positive effects on almost all other CSE elements, for instance agile practices are improved through experience reports or branching strategies from best practices.*

The model should open practitioners' eyes to new ideas when extending their CSE process. Furthermore, the relationships can start discussions for the future consolidation of the model: The categories *user*, *developer*, and *business* could be further summarized as *stakeholders*, while the categories *business*, *development*, and *operation* share common characteristics and might be combined as *BizDevOps*, following the naming conventions by Fitzgerald and Stol[2]. As highlighted, we see structural dependencies between *software management* and *knowledge* and between *quality* and *code*.

6.2 | Related Work

To the best of our knowledge, there are no previous studies with practitioners that address CSE as a process. Thus, we present and discuss studies that followed research approaches similar to our work and addressed specific CSE elements and CSE categories. Kuhrmann *et al.* research development approaches in practice[26]. They highlight the importance of traditional frameworks, a factor that is supported by our observation 12. Their results indicate that companies apply hybrid approaches, which are defined stepwise, in ways similar to the strategies we identified in Section 5.4. Mäkinen *et al.* report the widespread adoption of version control and continuous integration, based on semi-structured interviews with 18 organizations[27]. Our observations confirm this situation in practice. In fact, we found that most positive experience reports were stated in the respective CSE categories (Figure 5). Ståhl and Bosch interviewed practitioners to assess their experience of continuous integration and discuss benefits[28]. We can confirm most of their findings, e. g., that it increases developer productivity (observation 4). The same observation partially supports their finding that practitioners see improvements in project predictability. Further literature studies list benefits and challenges regarding continuous integration, delivery, and deployment[8,29]. Kevic *et al.* reveal the positive impact of experiments; this points toward a relationship between continuous deployment and a change in user behavior[30], and supports our model (Figure 6). Dybå and Dingsøyr highlight human and social factors in work settings that are similar to CSE[31]. Ayed *et al.* conclude that the success of agile practices depends on various social and inter-cultural factors[32]. Larusdottir *et al.* note the importance of teams' ability to trust in their capabilities[33]. Based on these reports and multiple answers in our study, we added the category *developer*, as described in Section 6.

7 | RESULTS ON HOW USAGE AND DECISION KNOWLEDGE CAN SUPPORT CSE (RQ2)

To present the results on RQ2, we provide a summary followed by a more detailed analysis for every research question. For RQ2.1, we analyzed the practitioners' attitude throughout the interview to derive their overall impression and their opinion on the overall feasibility, both grouped by *positive*, *neutral*, and *negative* responses. For RQ2.2 to RQ2.5, we analyzed the occurrence of the codes *usage knowledge focus*, *decision knowledge focus*, and *other focus* in relation to the answers to a research questions. During multiple interviews, we noticed that benefits were refined in a later question or used as an obstacle when viewing it from a different perspective. Further, answers to RQ2.4 and RQ2.5 were often very similar and dependent on how we phrased the questions during the semi-structured interview. We relied on a fine-grained distinction since the questions put different emphases: the extensions refer to the proposed CURES vision (Section 2.2) and do not assume major changes. The practitioners' responses regarding additions tend to result in fundamental changes to the CURES vision. Figure 7 provides an overview of the responses.

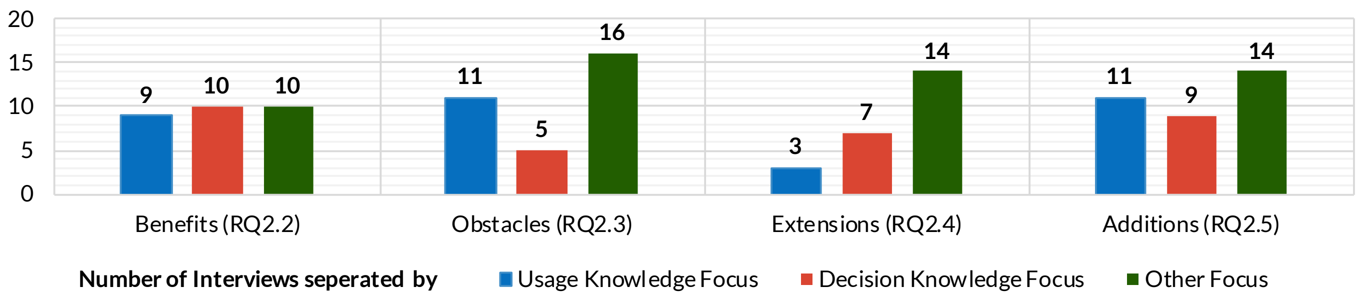


FIGURE 7 Overview of coded answers separated by research questions. A more detailed analysis of the responses is provided in Tables 2, 3, 4, and 5, in which we combined responses in case they were mentioned by at least two practitioners. Therefore, the numbers in this figure can be higher than the sum of responses per row in the tables. On the other hand, the numbers in this figure can be lower compared to the sum of rows in Tables 2, 3, 4, and 5, since this figure is based on interviews, while the tables detail the practitioners' responses, which allows a more fine-grained analysis.

7.1 | Practitioners’ Attitude toward the CURES Vision

The practitioners were asked to provide their overall impression and opinion toward the feasibility of the CURES vision to derive first assumptions about their attitude. We classified their response as *positive* if it clearly verbalized their support for one of the aspects. *Neutral* responses reflect responses that do not provide a definite point of view. A *negative* response articulates a substantial concern. The results are plotted in Figure 8.

RQ2.1 **What is the attitude of practitioners toward the CURES vision?** Practitioners primarily react positive to the integration of usage and decision knowledge into CSE. Regarding the practitioners’ overall impression, 15 positive, three neutral, and one negative responses were collected. For the overall feasibility, seven practitioners provide positive feedback, while eleven state a neutral- and two a negative opinion.

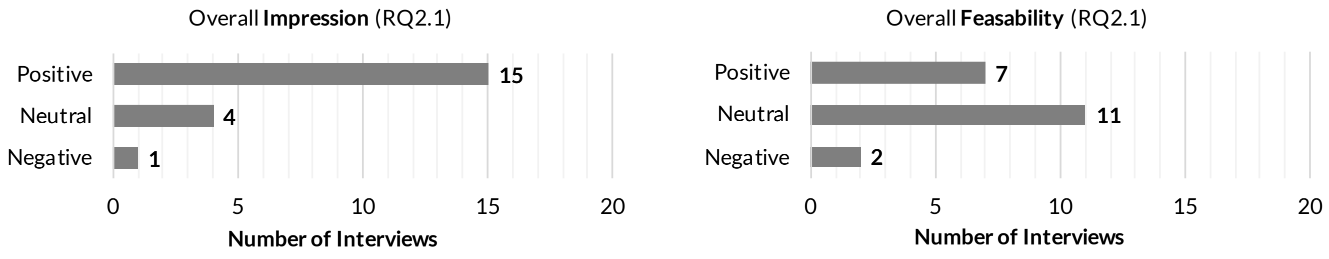


FIGURE 8 The practitioners’ attitude toward the CURES vision expressed by their overall impression and opinion on its feasibility.

In 15 out of 20 interviews, the practitioners noted a positive impression by describing the ideas as *beneficial* or *exciting*, as well as stressing out the advantages of a certain idea or aspect. Many of the practitioners acknowledge different aspects of the CURES vision, such as documenting decisions, as important and continued to refine their responses when discussing RQ2.2. During four interviews, the practitioners did not clearly take a stance for or against the proposed vision, however, they showed interest and did not articulate negative impressions. While generally being interested in the vision, one interviewee determined it impossible for their project at hand, rendering the vision only beneficial for specific projects; a closer elaboration on their reasoning is provided as part of RQ2.3, the *obstacles*, in which it is summarized under *Applicability*, i. e., finding themselves in a state in which multiple software versions are delivered to users.

With respect to the practitioners’ attitude toward the overall feasibility of the CURES vision, we noted that the practitioners provided highly subjective feedback dependent on their roles in the project, e. g., developers’ responses mainly addressed technical aspects. Out of the 20 interviews, we received seven positive responses, expressing that they can imagine the implementation of such a concept. Some of them noted that many elements of the vision have already been addressed by individual, detached systems, but given a vast amount of effort, they consider the vision to be feasible. With eleven interviews, the majority of practitioners took a neutral view on the vision’s feasibility by not clearly stating their opinion; some practitioners see challenges in the applicability in real-world scenarios, since the vision presented in Section 2.2 reflects a simplified scenario. A more detailed analysis of their responses is summarized under *Feasibility* in the *obstacles* discussion of Section 7.3. Two practitioners expressed a negative opinion regarding the overall feasibility of the CURES vision. They ground their doubts in technical- and process-oriented challenges.

7.2 | Practitioners’ Perceived Benefits of the CURES Vision

After an introduction to the CURES vision, we asked practitioners about benefits assuming that they would apply the vision in their company setting. We refined our questions more precisely around existing components, such as the arrangement of the *Dashboard* in Figure 1, in case the practitioners did not know where to start their observations. We consolidated their responses and found six major benefits as shown in Table 2.

RQ2.2 **What are benefits of the CURES vision perceived by practitioners?** From the practitioners’ responses, we derived six benefits: *Accountability* (7.2.1), *Traceability* (7.2.2), *Parallel Feedback* (7.2.3), *Automation* (7.2.4), *Flexibility*, and *Change Impact Analysis* (both in 7.2.5). Sixteen responses relate to decision knowledge and eleven for benefits summarized as *Accountability*. Usage knowledge-focused benefits are mentioned in 12 responses by the practitioners, while *Parallel Feedback* was highlighted four times. Nine responses focus on other benefits, most of them addressing *Traceability*.

TABLE 2 Benefits of the CURES vision sorted by the number of practitioners who addressed them. The practitioners' responses can have a usage knowledge, decision knowledge, or other focus. A star next to the name indicates benefits for which one response had more than one focus.

Benefit Name	**Accountability***	**Traceability***	**Parallel Feedback**	**Automation**	**Flexibility**	**Change Impact Analysis**
Usage Knowledge Focus	3	2	4	1	2	0
Decision Knowledge Focus	11	4	0	0	0	1
Other Focus	1	4	1	2	0	1
Number of Responses	12	8	5	3	2	2

7.2.1 | Accountability

The practitioners appreciate the possibility to discuss decisions collaboratively and to comment on the decisions made. The developers' possibility to capture their feedback and thoughts is seen as a key factor for success. In addition, documenting decisions within the development context, i. e., close to developers, and their easy retrievability is better than collecting the information in separate documents. Two practitioners explicitly highlight the benefit that the collected decision knowledge provides a proof on why a certain decision has been made, allowing to focus on the content of the decision, and not the individual—whether it is a member of the development team or a customer—that made the decision. The fact that decisions are collected continuously appeals to one practitioner who mentions that documenting is incorporated within the process of making the decision. One practitioner states that this can lead to good decisions on how the product should be further developed. Three practitioners stress accountability benefits by combining a usage and decision view on the development process: the user feedback supports the product management in decision-making regarding the next step of the product development. The *Dashboard* and the *Knowledge Repository* with integrated usage and decision knowledge, as introduced on Figure 1, are understood as a central point of interaction and discussion. The combination of the *Knowledge Repository* and the *Dashboard* becomes an important component during decision-making, almost as a generator for making decisions.

Observation 20 *Practitioners highlight that an explicit and structured way of decision-making improves transparency, comprehensibility, and replicability.*

7.2.2 | Traceability

Two practitioners praise the feature-based separation of knowledge. This allows to allocate knowledge directly to feature branches, enabling a development-driven documentation and simplifying the process of retrieving documented knowledge afterwards. Developers can track down knowledge right where it is created. In particular, traceability links to issue entries and deployed releases are automatically established. This enables to easily review the decisions made during particular software increments, a process that previously involved manual efforts. Three practitioners highlight that interweaving different components, such as the *Knowledge Repository* with the *CSE Infrastructure*, increases traceability. Three other practitioners mention practical aspects of traceability benefits that facilitate the daily workflows of developers: First, the CURES vision allows for an instant retrieval of typical coding issues and their solutions, e. g., regarding the design of an implemented method. Second, it reduces the need of using other tools, such as the version control system, which are cumbersome to use in order to figure out implementation decisions. Third, it allows to capture and explore decisions in a more intuitive way than in current decision documentation practices, such as when committing code.

Observation 21 *Feature branches as a main reference point simplify the traceability of usage and decision knowledge.*

7.2.3 | Parallel Feedback

The practitioners respond consistently that they perceive an environment in which they can evaluate two different implementations as highly beneficial for their development process. In this context, they emphasize the possibility to easily branch and merge the *better-performing* feature branch to continue their work. Two practitioners highlight similarities to A/B testing.

Observation 22 *Practitioners point out benefits in being able to contact users directly and receive instant feedback.*

7.2.4 | Automation

The practitioners mention benefits in the fact that the components support developers through semi-automated processes. For example, user feedback on an old release can be automatically mapped to the corresponding proposal, which allows the current development activities to focus only on relevant feedback. Furthermore, automating this process supports consistency and comparability of test results.

Observation 23 *Automation supports practitioners in synchronizing different knowledge artifacts.*

7.2.5 | Flexibility and Change Impact Analysis

Two practitioners highlight that CSE elements such as continuous delivery ease the process of deciding from whom usage feedback is to be collected. Similarly, with regard to change impact analysis, practitioners expect to retrieve more precise results on how a decision affects a software product. Based on these insights, a decision can be established, such as whether to continue improving a functionality that is used by a minority of users.

Observation 24 *The CURES vision can enable fine-grained usage and decision knowledge collection and assessment.*

7.3 | Practitioners' Perceived Obstacles of the CURES Vision

With respect to the CURES vision, we asked the practitioners for obstacles to its implementation within their project that go beyond and detail the feasibility and overall impression addressed in Section RQ2.1. We consolidated their responses and found six major obstacles as shown in Table 3.

RQ2.3 **What are obstacles of the CURES vision perceived by practitioners?** From the practitioners' responses, we derived six obstacles: *Feasibility* (7.3.1), *User Groups* (7.3.2), *Applicability* (7.3.3), *Cost* (7.3.4), *Usability* (7.3.5), and *Privacy* (7.3.6). With twelve responses for *Feasibility* and six both for *Applicability* and *Cost*, the practitioners see major obstacles in areas other than usage and decision knowledge. Furthermore, 15 responses relate to usage knowledge, from which more than half can be summarized under *User Groups*. None of the obstacles had a particular relation to decision knowledge.

TABLE 3 Obstacles to the CURES vision sorted by the number of practitioners who addressed them. The practitioners' responses can have a usage knowledge, decision knowledge, or other focus. A star next to the name indicates obstacles for which one response had more than one focus.

Obstacle Name	Feasibility	User Groups*	Applicability*	Cost	Usability	Privacy
Usage Knowledge Focus	0	9	3	0	1	2
Decision Knowledge Focus	0	0	0	0	0	0
Other Focus	12	1	6	6	3	1
Number of Responses	12	9	7	6	4	3

7.3.1 | Feasibility

One practitioner sees manual documentation as a major hurdle. Another practitioner stresses that especially developers prefer to perform their tasks efficiently, e. g., by using their system console, and are reluctant to use a different system than the one they are normally working with. Three practitioners see a problem in the strict separation between the *Knowledge Repository* and the *CSE Infrastructure*. One of them goes as far as suggesting to integrate the knowledge directly into the CSE infrastructure, which would benefit the portability of both, since references are less likely to get deprecated. However, they admit that the separation is reasonable, as long as they are physically located at a similar location. One practitioner criticizes that feedback items are related back to the master branch, not the feature branch at hand. Following the feasibility obstacles of the CURES vision, more sub-aspects with respect to the complexity were given by practitioners. They state that the many constraints between the developed software and its features cannot sufficiently be represented with the CURES vision. Furthermore, for some projects, the complexity increases given their dependability on other software projects or even physical devices. One practitioner finds a flaw in the simplified tree-structure indicated in Figure 1, since this model does not fit real-world applications, in particular if multiple individuals are involved over a period of time. One practitioner summarizes this aspect by stating that the CURES vision in its current state might be feasible for *off-the-shelf* software, but would struggle with complex *bespoke* software. Two practitioners are afraid that searching in a knowledge repository is challenging; this depends on the type of visualization. They propose different approaches to search for information and consider a text-based approach as the most difficult but also most important one.

Observation 25 *Practitioners are concerned about the required effort on their side to create and integrate documentation into the Knowledge Repository.*

7.3.2 | User Groups

Practitioners highlight that the user groups that provide user feedback tend to be internal development teams, not the targeted end user groups, which makes it difficult to use the received feedback as a representative baseline for decisions. They continue to highlight that—to correctly distinguish between two feature implementations—it requires many users to retrieve statistically significant results; this is barely reachable in such an environment. Two practitioners specify their concerns of finding users that provide usable feedback: First, it sometimes requires effort to understand the changes which reduces the users' acceptance to provide feedback. Second, the process of selecting users based on their skills is an important step, since it allows to ensure valuable feedback; however, this possibility is not provided with the current CURES vision.

Observation 26 *Practitioners prefer a system that can be used in a day-to-day workflow that is close to users, similar to an environment in which a few of their colleagues can briefly be consulted to receive feedback on a new software increment.*

7.3.3 | Applicability

Three practitioners see a high risk in providing different features to their user audience, since it would directly interfere with their business processes. For one practitioner, collecting usage data is a business-critical part of the product, which makes different versions of a software feature difficult, if not impossible to justify and bill. Two practitioners argue that the vision is incompatible with their way of developing and delivering software due to safety-critical aspects. Two practitioners mention technical problems in delivering multiple feature versions as demonstrated in the vision. While they are generally resolvable, it would require major efforts and adjustments in their development process to implement such changes. Finally, the practitioners mention a high coupling to server-side information which makes the applicability of the vision problematic. In particular, different staging environments that are used during the development make the supply of different client-side versions challenging. Further, even if this would be solved, repeating the process continuously poses new challenges.

Observation 27 *Practitioners are concerned about the effects on run-time aspects when creating different versions of a feature at the same time.*

7.3.4 | Cost

They see the overhead in developing multiple versions of a feature in parallel as an additional cost, however, they admit that the result would be improved as well. They highlight that the cost estimations can be difficult to justify to product managers or leaders. Further, two practitioners mention a cost aspect in ensuring that the user base regularly uses new feature additions. One practitioner points out that—in case they have two versions of a feature—they would rather sell this as an additional customization or variant of the feature. Likewise, one practitioner states that they would never develop a feature and then stop its development, nor would they evaluate three different alternatives against each other.

Observation 28 *According to practitioners, cost aspects of the CURES vision can be a major obstacle for its implementation.*

7.3.5 | Usability

Some practitioners wonder why a developer should use such a system, especially when they already have to deal with other, complex systems. Further, they are unsure whether the right stakeholders are addressed with the current version of the vision. This is the only time practitioners noted not enough roles as an obstacle. Two practitioners mention practical obstacles: while one practitioner notes that the proposed vision would solve a problem that they are currently not confronted with, another practitioner foresees upcoming problems in managing a new system by providing the *Dashboard* component, which needs to be integrated into user management systems and requires a variety of configuration possibilities.

Observation 29 *The integration of usage and decision knowledge into CSE must be easy to use to be adapted by stakeholders.*

7.3.6 | Privacy

The practitioners point out that the data that gets collected provides detailed insights into the user base. This may contain sensible information, which gets difficult to protect when processed and presented in other systems, such as the *Knowledge Repository* or *Dashboard*. One practitioner states that this obstacle can only be overcome if the development is clearly separated from production, which, however, interferes with the idea of continuous software engineering—following the quote *"You build it, you run it"*[34] the practitioner indicates that this obstacle needs to be addressed.

Observation 30 *Practitioners have privacy concerns when applying the CURES vision with real users. Therefore, the practitioners suggest to apply parts of the CURES vision in development environments only.*

7.4 | Practitioners' Extensions to the CURES Vision

We asked the practitioners for extensions to the presented CURES vision. In particular, we asked them to relate to their perceived benefits and how they could be strengthened. Likewise, we asked the practitioners to provide extensions to overcome obstacles and improve the overall feasibility of the CURES vision. Extensions are classified to be short-term changes to the CURES vision that do not require a major change and can be easily integrated into the existing concepts. We consolidated their responses and established six major extensions as shown in Table 4.

RQ2.4 **What are important extension to the CURES vision according to practitioners?** From the practitioners' responses, we derived six extensions: *Automation* (7.4.1), *Roles* (7.4.2), *Run-Time* (7.4.3), *Design-Time* (7.4.4), *Human and Processes* (7.4.5), and *Commit* (7.4.6). Multiple usage and decision knowledge-focused extensions are mentioned, however, with no noticeable peaks. In contrast, we found 27 responses that put a focus on other aspects, with many of them forming the topics *Automation*, *Roles*, and *Run-Time*.

TABLE 4 Extensions to the CURES vision sorted by the number of practitioners who addressed them. The practitioners' responses can have a usage knowledge, decision knowledge, or other focus. A star next to the name indicates extensions for which one response had more than one focus.

Extension Name	**Automation***	**Roles**	**Run-Time***	**Design-Time**	**Human & Processes***	**Commit**
Usage Knowledge Focus	1	2	2	0	1	0
Decision Knowledge Focus	1	0	0	2	2	2
Other Focus	8	6	6	3	4	0
Number of Responses	9	8	6	5	5	2

7.4.1 | Automation

The practitioners share a common idea for an extension to automate every aspect of the system. One practitioner formulates automation steps based on certain actions; for example, as soon as a feature branch is created, other processes such as creating a representation of this very feature branch in the *Knowledge Repository* should be triggered. They mention that it would be sufficient to have a basic support for this functionality, such as following naming conventions, in order to make it work. Another practitioner states that automation needs to be implemented differently for individual stakeholders and that a lowest common denominator should be found; otherwise, there are multiple things that can make such a system fail. As soon as one aspect that could be automated requires effort, such a system is difficult to establish; one practitioner justifies the need for automation with the fact that—if there is just one aspect that needs to be done manually—the data will turn inconsistent over time. Two practitioners highlight that automation allows them to see the relevant information at the time it is needed and enables to drill-down on information at hand.

Observation 31 *Practitioners request an additional component between the Knowledge Repository and the CSE Infrastructure that is in charge of executing various automation steps, such as showing relevant usage knowledge when available or synchronizing both components in case a change occurs.*

7.4.2 | Roles

Four practitioners state the need for adding capabilities for a project manager, i.e., a management role. According to their statements, these roles could benefit from the *Dashboard* by collecting information required for resource planning and estimation. At the same time, they can use it to be updated about latest developments, such as the delivery of certain software versions to users, or the availability of features in a feature branch. The *Dashboard* serves as an access point to information which previously required other tools, such as a git client, which some project manager might refrain from using. Two practitioners highlight different roles in the target audience, i. e., the users, and suggest to distinguish between internal users, such as developers, and external users. One practitioner sees the benefit of more roles in the fact that the more stakeholders are involved in making and capturing decisions, the more comprehensive and complete the knowledge will be. Another practitioner justifies the need for extended roles in being able to comply with privacy aspects; more roles allow for a more fine-grained selection of who is allowed to see which information.

Observation 32 *Practitioners stress the need for extending the CURES vision to more roles than developer and user.*

7.4.3 | Run-Time

With a focus on technical aspects, the practitioners express an interest in being provided with monitoring data from technical logs. This allows them to understand the features that are used by the users and control server-side capacities accordingly; based on metrics shown in the *Dashboard*, indications for roll-out or release dates could be inferred. One practitioner sees practical extensions that could be easily added to the current implementation: they would like to have an easy way to access older versions of a release, which could be integrated into the *Dashboard*. Another practitioner requests to see more dependencies to other software and physical devices in order to adjust the release of a feature at hand.

Observation 33 *Practitioners are interested in run-time knowledge that goes beyond usage knowledge to control operational aspects.*

7.4.4 | Design-Time

The practitioners demand an extension for an improved design-time support which goes beyond decision knowledge. They would like to add links to existing knowledge, such as architecture diagrams, guidelines, or ticket and issue discussions, which are not expressed in code and stored in other locations. Further, they would like to see a possibility to add meta-decisions independent of an individual feature branch.

Observation 34 *Being able to create links to design-time artifacts is important to practitioners.*

7.4.5 | Human and Processes

The practitioners highlight that that by applying the CURES vision, involved individuals need to rethink existing processes. They acknowledge that the *factor human* needs to be addressed. To convince individuals of processes, practitioners suggest to provide support such as acknowledging that—initially—more effort is required to make the process work. Likewise, they encourage the integration of process additions, such as making pull requests dependent on the availability of a certain type of knowledge, to foster the capturing of knowledge and to promote the usage of the system. One practitioner sees potential of the CURES vision in making aspects of the process visible, which might have been undiscovered before.

Observation 35 *The integration of usage and decision knowledge in CSE needs to be acknowledged by individuals and processes.*

7.4.6 | Commit

Two practitioners would like to see extensions in the way decision knowledge is captured. They proposed to be more verbose in commit messages. One developer is willing to apply a dedicated language or syntax to capture decisions knowledge when committing code—even if that means an additional workload at that stage. They are, however, not willing to review a previous commit after it has been added to the repository. One practitioner expresses their thoughts about a naive way of tagging decisions in commits—an idea, that we strive for with the CURES vision.

Observation 36 *Practitioners suggest to capture decision knowledge in documentation locations typical for CSE such as commit messages.*

7.5 | Practitioners' Additions to the CURES Vision

We asked the practitioners to list major feature additions, which they would expect to see from a system that allows to capture, maintain, and explore usage and decision knowledge during CSE. We strive for collecting long-term additions to the CURES vision that go beyond the presented idea and that require a major refinement of the concept. We consolidated their responses and established six major additions as shown in Table 5.

RQ2.5 **What are potential additions to the CURES vision according to practitioners?** From the practitioners' responses, we derived six additions: *Integration* (7.5.1), *Experimentation* (7.5.2), *Interaction and Reaction* (7.5.3), *Granularity* (7.5.4), *Community* (7.5.5), and *Developer Focus* (7.5.6). With eleven responses, usage knowledge additions stood out among the responses, with a peak in *Experimentation*. Five responses addressed additions regarding decision knowledge, most of them in *Integration*. The practitioners addressed additions of other areas in 14 responses, with a peak in *Interaction and Reaction*.

7.5.1 | Integration

The integration with tools available in a company's development process is stated most prominently by the practitioners: (1) support for external usage knowledge systems, i. e., to receive a greater variety of qualitative and quantitative feedback, (2) full support for functionality that is provided by the developers' integrated development environments, i. e., to reflect knowledge that is visualized in the dashboard immediately in code, (3) support for interfaces to systems that are close to hardware, i. e., to be able to read and write data via markdown or plain text, and (4) full support for any kind of project management tool. Overall, the integration should increase the stakeholders' acceptance by focusing on usability and efficiency.

TABLE 5 Additions to the CURES vision sorted by the number of practitioners who addressed them. The practitioners' responses can have a usage knowledge, decision knowledge, or other focus. A star next to the name indicates additions for which one response had more than one focus.

Addition Name	**Integration***	**Experimentation**	**Interaction & Reaction***	**Granularity***	**Community**	**Developer Focus**
Usage Knowledge Focus	3	4	1	1	0	2
Decision Knowledge Focus	3	0	0	2	0	0
Other Focus	5	1	4	1	3	0
Number of Responses	9	5	4	3	3	2

Observation 37 *Future concepts of usage and decision integration into CSE need to feature a high tool integration.*

7.5.2 | Experimentation

Five practitioners express specific feature additions: One technical leader describes scenarios in which users are required to find themselves in a particular setting to test a new feature addition; this setting is maybe given through a particular physical surrounding or a selection of parameters within the application. The practitioner would like to formalize such scenarios to be able to put users automatically into a setting in which they can use the feature. This, however, requires a high degree of integration, which makes its applicability for small features less likely. One practitioner would like to define experiments in a way that a continuous representation of results can be visualized, for example by a numeric value. Another practitioner evolves this idea by being able to measure any kind of interaction with the user to derive valuable information for the *Dashboard*. They suggest to observe time spans until a feature is understood by a user and try to derive phases in which the users find themselves—such as a learning phase, identification phase, orientation phase, searching phase, or productivity phase. Two practitioners focus their needs for additions in the interplay with users; they would like to address certain groups of users to specifically release a new feature to them. They propose that users can make the decision themselves, whether they want to use and provide feedback for a specific feature version.

Observation 38 *Collecting usage knowledge and providing monitoring and feedback components provides the opportunity for experimentation in CSE.*

7.5.3 | Interaction and Reaction

Adding more interaction possibilities to the *Dashboard* is seen as beneficial. The practitioners would like to control released versions from the *Dashboard*. Similarly, they request the possibility to connect and link existing knowledge to create new knowledge. Apart from the interaction aspects, the practitioners expect the *Dashboard* and its underlying *Knowledge Repository* to be *reactive*. For instance, the system should be able to send notifications to the stakeholders in case a previously defined threshold is reached. Likewise, relationships should be identified intelligently and presented to stakeholders by suggestions or warnings for interaction.

Observation 39 *Practitioners request to extend the functionality of the CURES dashboard beyond static representation of usage and decision knowledge.*

7.5.4 | Granularity

The practitioners would like to introduce more layers to the feature representation in the knowledge repository, in order to differentiate between knowledge for low-level code decisions, such as how to build a specific view, and knowledge regarding a large feature or use case. This would be similar to a differentiation known from *epics*, *user stories*, and *implementation tasks*. One practitioner states that in particular decisions require more information than just the decision itself; in particular, they asked for adding more meta information, such as who made a decision.

Observation 40 *Practitioners request major additions to the granularity of knowledge collected in the knowledge repository.*

7.5.5 | Community

The practitioners request a community component and justify their decision by referring to the constant availability of new technology and speed of development. According to their statements, some decisions have already been made and could be reused, in particular in the case of best practices for development decisions. Therefore, connecting the *Knowledge Repository* with external sources, such as *GitHub*, could be a valuable addition. One practitioner remarks that it would be a valuable addition if external standard repositories were treated differently, which would allow to rely on other stakeholders' knowledge, for example in the case of a version increment. An intermediate step is elaborated by a practitioner who expresses their thoughts for software that is developed within a consortium, which already crosses a company's borders. Allowing different *Dashboard* instances to communicate with each other is perceived as a beneficial addition.

Observation 41 *Learning from external knowledge can be a valuable addition to the internal body of knowledge during CSE.*

7.5.6 | Developer Focus

Since developers know how to trigger a certain behavior of the system, they should be able to simulate a user and produce data which is treated separately from the data produced by actual end users. An additional component could enable the replay of a situation which allows them to better understand a problem. Introducing this developer focus and separating it from other, end user mechanisms for usage monitoring would allow a more rigorous, yet effective way of collecting usage knowledge, without inferring with privacy aspects.

Observation 42 *The CURES vision can benefit from providing the ability to let developers assess their own behavior.*

8 | DISCUSSION ON AN IMPROVED CURES VISION (RQ2)

From the practitioners' responses, we received a broad view of how usage and decision knowledge can support them to improve the results of CSE. The assessment of RQ2.1 shows that the majority of practitioners have a positive overall impression of the CURES vision. Regarding their opinion on its feasibility, neutral responses dominated the practitioners' responses, followed by seven positive statements: Given the fact that most of the neutral responses can be related to either no clear statement or general remark, we also conclude the feasibility aspect as accomplished. We combined the practitioners' responses from RQ2.2 to RQ2.5 and updated our original CURES vision presented in Figure 1—the result is outlined in Figure 9 and detailed in the remaining part of this section, combined with two examples of how the improvements could be instantiated.

A major addition is represented by a new *Automation* component; multiple practitioners point out that the separation between the knowledge repository and the CSE infrastructure is too strict (see obstacles in Section 7.3: *Feasibility*). While the original CURES vision intended knowledge

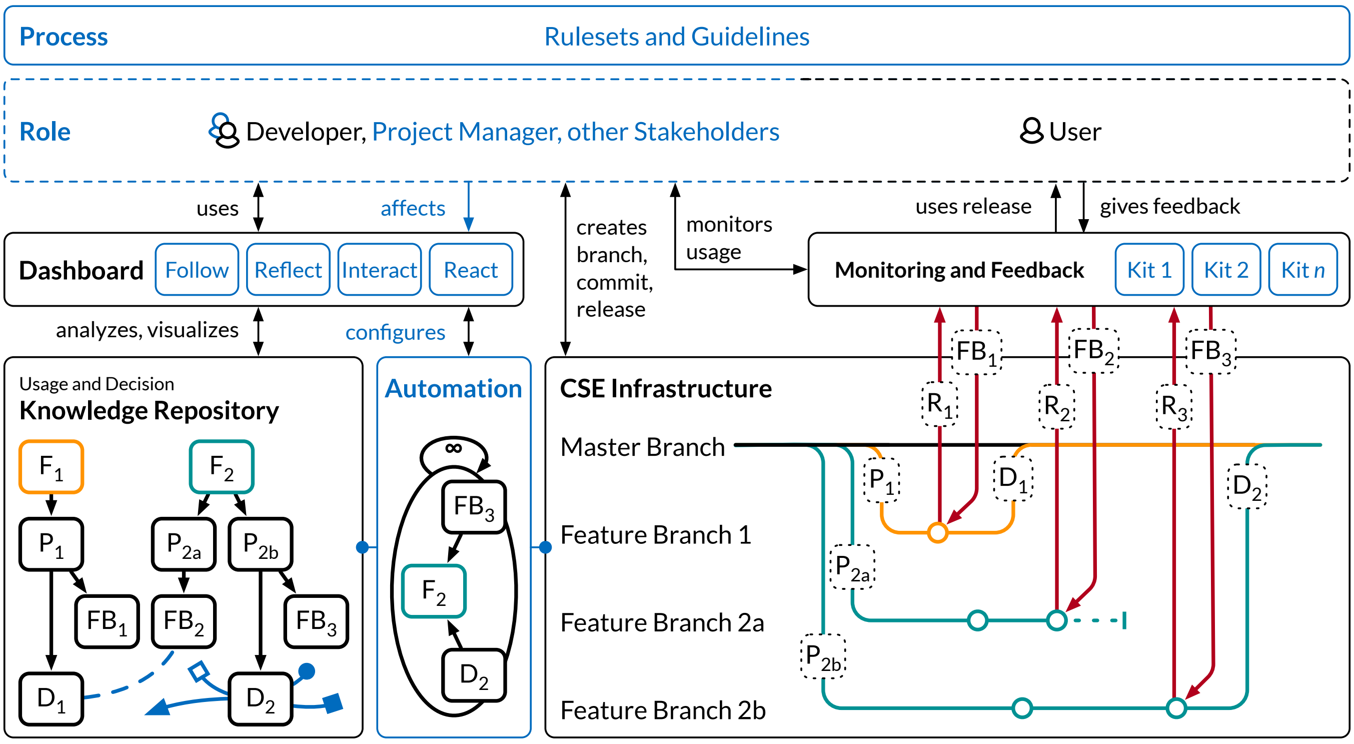


FIGURE 9 An improved version of the CURES vision based on the results of RQ2.2 to RQ2.5. Changes and additions to the version introduced in Section 1 are highlighted in blue color: major improvements can be found in a new *Automation* component, the addition of *Ruleset and Guideline* artifacts, as well as the extended functionality of the *Dashboard* and *Monitoring and Feedback* component. More *Stakeholders* are considered. Furthermore, the *Knowledge Repository* received refinements and the *CSE Infrastructure* minor changes.

elements to be transmitted between these components, the new automation component makes it clear that this needs to be done by an additional component. We see a broad field of tasks that can be accomplished by the automation component. Following the practitioners' suggestions, it can perform a set of tasks as soon as an event is triggered, such as collecting all available information for a feature branch that has just been created (see extensions in Section 7.4: *Automation*); this could also mean to collect other knowledge from technical logs (see extensions in Section 7.4: *Run-Time*). The automation component can also be in charge of regularly checking the knowledge available in the knowledge repository and investigating whether there are deprecated knowledge elements or missing knowledge links (see extensions in Section 7.4: *Automation*). It is also responsible for preparing knowledge that could be used within the new extensions of the dashboard, as explained in the following.

Example 3 *Knowledge originates from different sources, i.e., is created in heterogeneous platforms. For instance, the CSE infrastructure might be an online repository server that is running independently from a server that hosts the knowledge repository or the monitoring and feedback component. A hook architecture can enable the connection aspects which are required to achieve the goals of the automation component. Web hooks represent a simple way of connecting sources, since they usually authenticate only via a web address and a secret; this makes them easy to integrate into already existing systems. Furthermore, they are event-based, which fits the requirements for the automation component: whenever a new commit is pushed, the knowledge repository can receive a notification from the CSE infrastructure. Based on this, it can start gathering data. Likewise, whenever a developer captures a decision knowledge or the user creates a new feedback, this knowledge can be pushed into the knowledge repository and automatically related to a commit.*

The automation component is further linked to the *Dashboard*, which was addressed by the practitioners as part of multiple responses. Initially, the dashboard's intention was to *consume* knowledge that is stored in the knowledge repository—now represented by the *Follow* and *Reflect* components. With the practitioners' feedback, we find it important to be more specific about the possibilities the dashboard offers [10]: The *Interact* and *React* components make the dashboard more active, offering the possibility to inform stakeholders about relevant changes (see additions in Section 7.5: *Interaction and Reaction*). A new aspect is given through the *affect* relationship from the role toward the dashboard; depending on *who* is using the dashboard, it changes its appearance and the way it works to encourage knowledge gain from stakeholders (see extensions in Section 7.4: *Roles*).

Several practitioners address the limited set of *Roles* in the original version of the CURES vision (see obstacles in Section 7.3: *Usability* and extensions in Section 7.4: *Roles*). Therefore, we extend the scope of roles, considering any possible stakeholder with an interest in the product development. We see *Cost* (see obstacles in Section 7.3) as a motivation to explicitly mention *Project Manager* next to *Developer*. Furthermore, by fusing the user role into the other stakeholder roles, we acknowledge that our CURES vision is planned to be used in a development-close environment, in which a developer is the first *user*, followed by other stakeholders, of which one of them might represent the actual end user. This removes uncertainty regarding the *User Groups* (see obstacles in Section 7.3), i. e., the availability of end users, their necessity to have a certain level of knowledge and to invest effort. Also, it increases the *Applicability* (see obstacles in Section 7.3), since—if performed by a developer—business- and safety-critical areas of a software product can be tested in a controlled environment. Finally, having developers act as users can reduce concerns regarding *Privacy* (see obstacles in Section 7.3), since they are interacting with the application while they know that they are being observed. However, this might influence the results. Moreover, it accommodates the request of developers for a more *Developer-focused* (see extensions in Section 7.4) system.

We improve the capabilities of the *Knowledge Repository*. First, we follow the suggestion to link more sources of knowledge to existing elements in our knowledge graph; as indicated with $\mathtt{D}_2$, different kinds of knowledge, such as *Run-Time* (see extensions in Section 7.4) or *Design-Time* (see extensions in Section 7.4) knowledge can be linked to the decision. Second, links between knowledge elements, such as between $\mathtt{FB}_2$ and $\mathtt{D}_1$, can help to address the *Feasibility* (see obstacles in Section 7.3), and potentially also contribute to the aspects of *Granularity* (see additions in Section 7.5).

Supported by the practitioners' reports (see benefits in Section 7.2: *Parallel Feedback*), we enhanced the composition of the *Monitoring and Feedback* component. By the introduction of *Kit* components, different modules for user *Experimentation* (see additions in Section 7.5) can be added to the monitoring and feedback component. In the case of a special knowledge need, a kit can be activated and used for collecting usage data; this might even be the *Integration* (see additions in Section 7.5) of an external service that offers specialized monitoring features.

Example 4 *Stakeholders might have different interests about the usage knowledge that is collected. This need is reflected in the possibility to use kits. For example, a FeatureKit addition allows stakeholders to analyze the execution status of newly developed features increments [13]; information whether a feature was started, completed, or canceled helps to understand the usability of a feature. Other kits can provide additional data, such as information about the users' behavior or their impression that was derived from a hardware or a software sensor. By combining these knowledge sources, the decision-making on the next steps for the feature increment can be supported or improved.*

A minor addition lies in the introduction of a new *Process* layer, which should ensure the *Human and Processes* (see extensions in Section 7.4) aspects, which are required to successfully implement the CURES vision. This may be reflected in the introduction of a basic naming convention (see extensions in Section 7.4: *Automation*) to support the automation aspects of the CURES vision. Likewise, it could be understood as a first step toward a *Community* (see additions in Section 7.5) connection which would mean to open up knowledge to other parties. Furthermore, minor changes to the *CSE Infrastructure* encompass the routing of feedback to the commit from which it has been originated (see obstacles in Section 7.3: *Feasibility*).

9 | CONCLUSION

We describe the method and observations of a semi-structured interview study involving 24 practitioners from 17 companies during 20 interviews. The study provides an overview of the current state of practice in CSE, as a way to assist practitioners in understanding CSE. Further, the study results in a refined approach to integrate usage and decision knowledge into CSE according to the practitioners' feedback on its initial version.

Regarding RQ1, i. e., how companies apply CSE, we found that the practitioners have different perspectives, i. e., *tool*, *methodology*, *developer*, *life cycle*, and *product management* perspectives. Most relevant elements of CSE are *user* and *team commitment*, as well as a high degree of maturity of *automated loops*. With respect to CSE elements, the practitioners report more positive experiences than negative ones, but more than half of the responses were neutral. This might indicate that many practitioners are currently only testing various CSE element in the field. For the future, the practitioners focus on three strategies: *enhancement*, *expansion*, and *on-demand adaption*. To support practitioners in their approaches to establish CSE in companies, we created the *Eye of CSE* model based on our interview observations. The *Eye of CSE* allows practitioners to identify relevant CSE elements, the mutual relationships between them, and to devise future additions to their own projects. We do not claim that the model is complete.

Regarding RQ2, i. e., how usage and decision knowledge support CSE, we found that most of the practitioners have a positive impression of the proposed CURES vision, and—while more than half of the responses regarding its feasibility were neutral—there is the tendency that the practitioners consider the CURES vision as feasible. The practitioners mention six different areas of benefits of which *Accountability*, *Traceability*, and *Parallel Feedback* stood out in the number of occurrences. All of them relate to usage and decision knowledge, which strengthens the overall CURES vision. At the same time, the practitioners were primarily concerned about the *Feasibility*, *Dependability*, and *Applicability*. In particular, they highlighted technical challenges and the dependability on users as major problems that could arise when implementing the CURES vision. The practitioners provided insights on extensions and additions they perceive as most important for the CURES vision: *Automation*, *Roles*, and *Run-Time*, as well as *Integration*, *Experimentation*, and *Interaction and Reaction*. We revised our initial idea of the CURES vision and created an improved version.

Notably, results from RQ1 are reflected within RQ2. For instance, the importance of the users' commitment as stated in observation 8 is reflected in the addition of the *Process* component in the improved CURES vision to highlight the need for a shared ruleset and guidelines. Likewise, the observation 7 is reflected in the practitioners' responses with respect to the *Dependability* (see obstacles in Section 7.3) on users; as a consequence, we refined the notion of a user and extended the target audience of the CURES vision by more stakeholders.

We see a continuation of the study in the form of a survey to further validate the perception of the *Eye of CSE*. In particular, it should be evaluated whether a large number of practitioners would agree with our allocation of CSE elements to categories. This allows to further refine the position of individual CSE element within the Eye of CSE and thereby unfold more relationships between the CSE elements. When practitioners use the *Eye of CSE* as suggested as a checklist, they would benefit from this knowledge. Furthermore, insights from practitioners from different domains might open up new research directions. In addition, we are working on the implementation of the presented CURES vision. We are incrementally implementing the proposed components in the form of tool [17] and platform [35] support. Using this work in progress, we are evaluating the acceptance of the implementations by applying them in a multi-project course that provides access to a real-world development setting [36,37,38]. We strive to receive further insights which we use for improvements.

ACKNOWLEDGMENTS

This work was supported by the DFG (German Research Foundation) under the Priority Programme SPP1593: Design For Future – Managed Software Evolution (CURES project). We would like to thank the practitioners for their participation in the interviews and sharing their insights, as well as Rana Alkadhi and Doris Keidel-Mueller for their valuable feedback.

References

1. Bosch Jan. *Continuous Software Engineering: An Introduction*. Cham: Springer; 2014.

2. Fitzgerald Brian, Stol Klaas-Jan. Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software*. 2017;123:176–189.

3. Humphrey Watts S.. The Software Engineering Process: Definition and Scope. *SIGSOFT Software Engineering Notes*. 1988;14(4):82–83.

4. Krusche Stephan, Bruegge Bernd. CSEPM - A Continuous Software Engineering Process Metamodel. In: 2017 IEEE/ACM 3rd International Workshop on Rapid Continuous Software Engineering (RCoSE):2-8; 2017; Buenos Aires, Argentina.

5. Olsson Helena Holmström, Alahyari Hiva, Bosch Jan. Climbing the "Stairway to Heaven" – A Mulitiple-Case Study Exploring Barriers in the Transition from Agile Development towards Continuous Deployment of Software. In: 2012 38th Euromicro Conference on Software Engineering and Advanced Applications:392-399; 2012; Washington, DC, USA.

6. Ståhl Daniel, Mårtensson Torvald, Bosch Jan. The continuity of continuous integration: Correlations and consequences. *Journal of Systems and Software.* 2017;127:150–167.

7. Rahman Akond Ashfaque Ur, Helms Eric, Williams Laurie, Parnin Chris. Synthesizing Continuous Deployment Practices Used in Software Development. In: 2015 Agile Conference:1-10; 2015; Washington, DC, USA.

8. Rodríguez Pilar, Haghighatkhah Alireza, Lwakatare Lucy Ellen, et al. Continuous deployment of software intensive products and services: A systematic mapping study. *Journal of Systems and Software.* 2017;123.

9. Johanssen Jan Ole, Kleebaum Anja, Bruegge Bernd, Paech Barbara. Towards a Systematic Approach to Integrate Usage and Decision Knowledge in Continuous Software Engineering. In: 2nd Workshop on Continuous Software Engineering:7–11; 2017; Hannover, Germany.

10. Johanssen Jan Ole, Kleebaum Anja, Bruegge Bernd, Paech Barbara. Towards the Visualization of Usage and Decision Knowledge in Continuous Software Engineering. In: 2017 IEEE Working Conference on Software Visualization (VISSOFT):104-108; 2017; Shanghai, China.

11. Johanssen Jan Ole, Kleebaum Anja, Paech Barbara, Bruegge Bernd. Practitioners' Eye on Continuous Software Engineering: An Interview Study. In: Proceedings of the 2018 International Conference on Software and System Process:41–50ACM; 2018; Gothenburg, Sweden.

12. Maalej Walid, Happel Hans-Jörg, Rashid Asarnusch. When Users Become Collaborators: Towards Continuous and Context-aware User Input. In: Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications:981–990ACM; 2009; Orlando, Florida, USA.

13. Johanssen Jan Ole, Kleebaum Anja, Bruegge Bernd, Paech Barbara. Feature Crumbs: Adapting Usage Monitoring to Continuous Software Engineering. In: Kuhrmann Marco, Schneider Kurt, Pfahl Dietmar, et al. , eds. *Product-Focused Software Process Improvement*, :263–271Springer International Publishing; 2018; Cham.

14. Seffah Ahmed, Metzker Eduard. The Obstacles and Myths of Usability and Software Engineering. *Communications of the ACM.* 2004;47(12):71–76.

15. Dutoit Allen H, McCall Raymond, Mistrík Ivan, Paech Barbara. *Rationale Management in Software Engineering*. Berlin, Heidelberg: Springer; 2006.

16. Alexeeva Zoya, Perez-Palacin Diego, Mirandola Raffaela. Design Decision Documentation: A Literature Overview. In: LNCS, vol. 5292: Cham: Springer Berlin Heidelberg 2016 (pp. 84–101).

17. Kleebaum Anja, Johanssen Jan Ole, Paech Barbara, Bruegge Bernd. Tool Support for Decision and Usage Knowledge in Continuous Software Engineering. In: Krusche Stephan, Lichter Horst, Riehle Dirk, Steffens Andreas, eds. *Proceedings of the 3rd Workshop on Continuous Software Engineering*, :74–77CEUR-WS.org; 2018; Ulm, Germany.

18. Kleebaum Anja, Johanssen Jan Ole, Paech Barbara, Alkadhi Rana, Bruegge Bernd. Decision Knowledge Triggers in Continuous Software Engineering. In: Proceedings of the 4th International Workshop on Rapid Continuous Software Engineering:23–26ACM; 2018; Gothenburg, Sweden.

19. Brunet João, Murphy Gail C, Terra Ricardo, Figueiredo Jorge, Serey Dalton. Do developers discuss design?. In: Proceedings of the 11th Working Conference on Mining Software Repositories - MSR 2014:340–343ACM Press; 2014; Hyderabad, India.

20. Krusche Stephan, Alperowitz Lukas, Bruegge Bernd, Wagner Martin O.. Rugby: An Agile Process Model Based on Continuous Delivery. In: Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering:42–50; 2014; Hyderabad, India.

21. Krusche Stephan. Rugby - A Process Model for Continuous Software Engineering. PhD thesisTechnical University Munich, Germany2016.

22. Myers Michael D, Newman Michael. The Qualitative Interview in IS Research: Examining the Craft. *Information and Organization.* 2007;17(1):2–26.

23. Runeson Per, Host Martin, Rainer Austen, Regnell Björn. *Case Study Research in Software Engineering: Guidelines and Examples*. Hoboken, N.J.: John Wiley & Sons; 2012.

24. Saldaña Johnny. *The Coding Manual for Qualitative Researchers.* Los Angeles ; London ; New Delhi ; Singapore ; Washington DC: SAGE Publications; 2 ed.2009.

25. Torchiano Marco, Ricca Filippo. Six reasons for rejecting an industrial survey paper. In: 2013 1st International Workshop on Conducting Empirical Studies in Industry (CESI):21-26; 2013; San Francisco, CA, USA.

26. Kuhrmann Marco, Diebold Philipp, Münch Jürgen, et al. Hybrid Software and System Development in Practice: Waterfall, Scrum, and Beyond. In: Proceedings of the 2017 International Conference on Software and System Process:30–39ACM; 2017; Paris, France.

27. Mäkinen Simo, Leppänen Marko, Kilamo Terhi, et al. Improving the delivery cycle: A multiple-case study of the toolchains in Finnish software intensive enterprises. *Information and Software Technology.* 2016;80:175 - 194.

28. Ståhl Daniel, Bosch Jan. Experienced Benefits of Continuous Integration in Industry Software Product Development: A Case Study. In: Proceedings of the 12th IASTED International Conference on Software Engineering, SE 2013:736–743; 2013; Innsbruck, Austria.

29. Shahin Mojtaba, Ali Babar Muhammad, Zhu Liming. Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. *IEEE Access.* 2017;5(Ci):3909–3943.

30. Kevic Katja, Murphy Brendan, Williams Laurie, Beckmann Jennifer. Characterizing Experimentation in Continuous Deployment: A Case Study on Bing. In: Proceedings of the 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP):123-132; 2017; Buenos Aires, Argentina.

31. Dybå Tore, Dingsøyr Torgeir. Empirical studies of agile software development: A systematic review. *Information and Software Technology.* 2008;50(9-10):833–859.

32. Ayed Hajer, Vanderose Benoit, Habra Naji. Agile cultural challenges in Europe and Asia: insights from practitioners. In: Proceedings of the 39th International Conference on Software Engineering: SEIP Track:153–162IEEE; 2017; Buenos Aires, Argentina.

33. Larusdottir Marta, Gulliksen Jan, Cajander Åsa. A license to kill – Improving UCSD in Agile development. *Journal of Systems and Software.* 2017;123:214–222.

34. O'Hanlon Charlene. A Conversation with Werner Vogels. *Queue.* 2006;4(4):14:14–14:22.

35. Johanssen Jan Ole. Continuous User Understanding for the Evolution of Interactive Systems. In: Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '18):15:1–15:6ACM; 2018; Paris, France.

36. Johanssen Jan Ole, Henze Dominic, Bruegge Bernd. A Syllabus for Usability Engineering in Multi-Project Courses. In: 16. Workshop Software Engineering im Unterricht der Hochschulen (SEUH):126–133; 2019; Bremerhaven, Germany.

37. Kleebaum Anja, Johanssen Jan Ole, Paech Barbara, Bruegge Bernd. Teaching Rationale Management in Agile Project Courses. In: 16. Workshop Software Engineering im Unterricht der Hochschulen (SEUH):134–145; 2019; Bremerhaven, Germany.

38. Bruegge Bernd, Krusche Stephan, Alperowitz Lukas. Software Engineering Project Courses with Industrial Clients. *ACM Transactions on Computing Education.* 2015;15(4):17:1–17:31.

How to cite this article: Johanssen JO, Kleebaum A, Paech B, and Bruegge B. Continuous software engineering and its support by usage and decision knowledge: An interview study with practitioners. *J Softw Evol Proc.* 2019;e2169. https://doi.org/10.1002/smr.2169