How do Practitioners Manage Decision Knowledge during Continuous Software Engineering?

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

[1]Heidelberg University, Heidelberg, Germany, {kleebaum, paech}@informatik.uni-heidelberg.de
[2]Technical University of Munich, Munich, Germany, {jan.johanssen, bruegge}@in.tum.de

***Abstract*—Continuous software engineering (CSE) is an agile process that supports lightweight, flexible, and rapid software development. Decision-making is crucial for CSE, and developers need to know the decisions made and the related rationale to evolve the software. This knowledge is called decision knowledge. The management of decision knowledge in CSE environments remains unexplored. The agile manifesto suggests to value working software over comprehensive documentation as well as individuals and interactions over processes and tools. What does this mean for the documentation, exploitation, and sharing of decision knowledge? We report on results from an interview study with 24 practitioners from 17 companies on how decision knowledge is managed during CSE. The practitioners mainly capture decision knowledge in an informal way, for example, in natural language discussions. Wiki and issue tracking systems represent the preferred medium to preserve decision knowledge. Mentioned benefits are an improved decision-making process, accountability, knowledge sharing, and reuse. However, the exploitation of the captured decision knowledge remains partly unclear.**

***Index Terms*—Decision knowledge, rationale, interview study, continuous software engineering, agile software development.**

I. Introduction

Continuous software engineering (CSE) is an agile process that supports lightweight, flexible, and rapid software development [1]. This process is intertwined with ongoing issue-solving and decision-making. For example, developers make decisions regarding the software development process, existence and non-existence of software artifacts, or the software quality [2]. Developers need decision knowledge, i. e., knowledge about the decisions and their rationale, to evolve the software. Rationale covers the justification behind decisions.

CSE offers opportunities and bears challenges for the management of decision knowledge. The opportunities are that developers already document decision knowledge in documentation locations such as commit messages, issue comments, or pull requests during established development practices, e. g., when committing code changes. Thus, the documentation of decision knowledge during CSE is non-intrusive in comparison to using a separate tool. This might help to overcome the capture problem that is often mentioned in articles about decision management [3]. The challenges are that the documented decision knowledge might be hard to access and exploit for developers since it is distributed and might be not formalized. Further, decisions can rapidly be changed which might lead to an inconsistent and outdated documentation. It is unexplored how decision knowledge is managed during CSE in practice.

We conducted a semi-structured interview study with practitioners from 17 companies using CSE. We already reported on the state-of-the-practice of CSE in these companies [4], [5]. In this paper, we report on how the companies manage decision knowledge. We contribute insights on which types of decisions practitioners think are important to capture, how they capture decision knowledge, what benefits they see in capturing decision knowledge, how they share decision knowledge, and how they deal with change. These insights should help us to reach our overall goal to support software evolution with decision and usage knowledge in CSE [6], [7].

The remainder of this paper is structured as follows. In Section II, we present our research questions, the procedure of the interview study, and descriptive data of the interviewees. In Section III, we answer the research questions. Section IV discusses this work and Section V lists threats to validity. Section VI presents related work on decision knowledge management in practice. Section VII concludes the paper.

II. Research Method

In the following, we describe our research questions, the interview study, and data about the study participants.

A. Research Questions

We focus on four research questions, each refined by sub-questions that we asked the practitioners during the interviews.

RQ1: *Which decisions are captured, why, and how?* This research question investigates approaches to explicitly capture decisions and rationale during CSE within companies. Sub-questions are: Which types of decisions do practitioners capture? Where do practitioners capture the decisions, with which techniques and tools? Do practitioners link decisions and rationale to other software artifacts and if so, how? How do practitioners preserve the evolution history of decisions? When and how often do practitioners capture decisions? Why do practitioners capture decisions, i. e., what are the benefits and what do they do with the captured decisions?

RQ2: *Which important decisions are not captured by practitioners, why not?* This research question aims to find types of decisions that remain implicit and reasons why they are not captured. Sub-questions are: Which important decisions do practitioners not capture during CSE? Why do practitioners not capture these decisions? What would be the benefits if practitioners captured these decisions?

DOI reference number: 10.18293/SEKE2019-206

RQ3: *How do practitioners share decision knowledge?* We want to investigate how practitioners share decision knowledge during CSE, with these sub-questions: What are the knowledge sources from which practitioners retrieve necessary information for decisions that are not captured? How do practitioners share knowledge to avoid knowledge vaporization?

RQ4: *How do practitioners deal with changing decisions?* We aim to investigate on how practitioners deal with change, with the sub-question: How do practitioners identify parts of the system that are affected by new or changed decisions?

B. Interview Study

In the following, we summarize the realization of the interview study that is described in more detail in [4], [5].

1) Procedure: We performed a semi-structured interview study [8], which is a survey study and thus a means to perform a field study [9], [10]. We separated the study into a *design and planning*, *data collection*, and *data analysis* phase.

During the *design and planning* phase, we prepared a questionnaire. Its first part addresses the practitioners' background and working context. Furthermore, it contained interview questions as listed below the respective research question in Section II-A. The interviews also included research questions that are not addressed in this article (see [4], [5]). We contacted companies which to our knowledge apply CSE.

During the *data collection* phase, we conducted 20 interviews between April and September of 2017. The interviews were conducted either in person or via phone. 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. We guaranteed the anonymity of the practitioners by publishing only aggregated results.

In the *data analysis* phase, we analyzed the transcripts [11]. We utilized a *qualitative data analysis* software to apply two stages. During the first stage, we allocated answers to an interview question. Hereafter, we performed a fine-grained coding stage. To answer the interview questions concerning the types of decisions captured and not captured, we derived the codes from Kruchten's taxonomy [2]. This taxonomy distinguishes between existence decisions, non-existence decisions (bans), property decisions, and executive decisions. For the remaining interview questions, we identified emerging topics and coded the answers in terms of these topics. We analyzed the results quantitatively. In the case that two interviewees participated in an interview, we treated their answers as one subject.

2) Participants: During 20 interviews, we interviewed 24 practitioners from 17 companies. While seven of the companies provide consultancy services, ten companies develop software products. Based on their role description, we grouped the 24 practitioners into five categories: *CSE specialists* (5), e. g., a continuous deployment manager or a DevOps engineer, *developers* (6), *project managers* (6), *technical leaders* (6), and one *executive director*. On average, the practitioners have spent two years in the respective role, have an experience in IT projects of ten years, and participated in 19 IT projects.

III. Results

We present results on the research questions introduced in Section II-A. Each subsection starts with a summary followed by a more detailed analysis of the research question.

A. Decisions Captured during CSE

In three interviews, the practitioners state that they do not capture decisions at all. In these cases, we skipped the interview questions for RQ1 and started with RQ2.

> **RQ1**: *Which decisions are captured, why, and how?* Practitioners mainly capture executive and existence decisions regarding the software architecture and feature implementation. They mostly capture decisions in wiki and issue tracking systems in informal discussions and rely on techniques for establishing trace links and version control that come with these systems. Practitioners capture decisions as part of regular practices, such as code reviews and meetings. They mention improved decision-making, accountability, knowledge sharing, as well as reuse support as benefits. However, the exploitation of the decision knowledge is partly unclear.

1) Types of Captured Decisions: Twelve practitioners report that they capture *executive decisions*, i. e., decisions concerning the software development process, technologies, or applied tools. Such decisions impact the entire project or several projects. Similar to non-CSE environments, the executive decisions can be made by a steering committee. However, one practitioner highlights that CSE supports strongly that developers themselves are enabled to make high-level decisions. As examples, the practitioners mention to capture the decision to use a certain branching strategy or to do continuous delivery. One practitioner mentions to capture decisions on when they can consider a task as done, i. e., the definition of done, and on when a build can be deployed to the users. *Existence decisions* state that some elements will appear in the software [2]. Thirteen practitioners capture existence decisions concerning requirements, architecture, implementation, test cases, and bug reports. Six practitioners report that they capture decisions related to the elicitation, prioritization, and effort estimation of requirements for features. Eight practitioners mention that they capture architecture decisions and another nine mention that they capture decisions regarding the implementation of features, e. g., on why a class was created. *Non-existence decisions or bans* state that some elements will not appear in the software [2]. Five practitioners report to capture possible alternatives to solve a decision problem during their decision-making process. After evaluating the alternatives against criteria, they pick one alternative as the decision. The alternatives they discard are documented non-existence decisions. One practitioner reports to capture decisions regarding the prioritization of test cases and bug fixing activities based on risk assessment. *Property decisions* concern the quality of the system and can be guidelines, design rules, or constraints [2]. One practitioner provides the example that they captured the decision on how to deal with data inconsistency after they have replaced their relational database with a NoSQL database.

2) Documentation Locations, Techniques, and Tools: Practitioners use various documentation locations, techniques, and tools to capture decisions during CSE. Eight practitioners mention that they capture decisions in *external documents and tools* such as *Word* files, architecture design documents, or final project reports. Only one practitioner mention to use an architecture management tool, which in their case is the *Enterprise Architect*. Thirteen practitioners report on using a *wiki system* such as Confluence. One practitioner mentions that they rely on template pages to capture decisions. Ten practitioners mention that they capture decisions in an *issue tracking and project management system*, such as *JIRA* or *Redmine*, as part of the issue description and its comments. One practitioner describes that they use a distinct discovery issue type to indicate that a decision needs to be made. Similarly, another practitioner mentions that they use a tag to mark those issues that contain an open decision. One practitioner highlights that in their opinion *pull requests* are the best place to capture decisions to implement features. They create feature branches for a requirement and create a pull request directly afterwards to discuss the feature implementation within the pull request. Another practitioner reports that they document decisions as part of the *code* in comments and in *code reviews*. Code reviews can be done in pull requests, issue comments, or using dedicated code review systems, such as *gerrit*. Three practitioners mention commit messages as a documentation location for decisions and another three mention informal communication systems, e. g., *chat tools* like *Slack*, and *emails*.

3) Linked Artifacts: None of the practitioners uses a particular technique to establish links between captured decision knowledge and software artifacts. However, practitioners mention techniques that come naturally with capturing decision knowledge in some documentation locations. For example, the practitioners report that the decisions captured in the issue tracking system can be traced to the respective issues such as user stories and also to artifacts that are linked to these issues, e. g., software components and code. In addition, they also mention that separate documents or wiki pages can be tagged; for instance, version numbers can enable traceability between decisions and software builds. Practitioners find it hard to keep the documentation of decisions and software artifacts in a consistent state. The practitioner using the architecture knowledge management tool criticizes that there are no links between the design models and the wiki system where they also capture decisions. They insert snapshots of the models into the wiki page, which they rate as highly unusable, especially when the models get changed. Another practitioner suggests to capture decisions as close to the code as possible.

4) Evolution History of Decisions: Similar to the linking of decision knowledge and artifacts, a preservation of the evolution history comes naturally in those *documentation locations that offer version control*, e. g., the issue tracking system. One practitioner describes that they have a technique to *mark a revised decision*; they link the revised decision with the new one rather than overwriting the revised decision. However, the practitioner admits that they never used the technique.

5) Capturing Practices and Frequencies: Six practitioners report that they mainly capture decisions *on demand*, e. g., when planning bigger updates. One practitioner states that they only capture a decision in case they need to discuss on it, i. e., for controversial issues. Seven practitioners mention that they capture decisions as part of *regular practices* such as code reviews, meeting, and retrospectives. The practitioner reporting about the tag to mark an open decision states that the product owner regularly filters for such tagged issues.

6) Benefits and Exploitation: Five practitioners mention that they document decisions since the documentation improves *decision-making*, i. e., leads to better decisions since the criteria become clearer. Eight practitioners state that they capture decisions and rationale for *accountability* reasons, e. g., as a proof on why a certain feature has been developed and to avoid misunderstandings in the future. One of them states to exploit captured decisions when recovering a former state of the software. Three practitioners state that they capture decisions for *knowledge sharing* purposes. Among them, one practitioner highlights that it is necessary to share the knowledge about where a new decision needs to be made, i. e., also to share issues. Two practitioners capture decisions in order to support *reuse* in the future to avoid duplicated work.

We asked the practitioners to rate the statement *The explicit capturing of decisions benefits our software development process* with one answer from a five point Likert scale. In thirteen interviews the practitioners rated this statement: one disagreed, three were neutral, and nine agreed (Figure 1). The practitioners who disagreed and were neutral emphasized that if the utilization of the captured knowledge was more clear, they would give a higher rating.

B. Decisions not Captured during CSE

Although some practitioners capture executive, existence, non-existence, and property decisions during CSE, others either a) do not capture the same type of decisions or b) provide other concrete examples for decisions that they do not capture.

> **RQ2**: *Which important decisions are not captured by practitioners, why not?* Decisions regarding the CSE process, prioritization, alternatives that are not selected (non-existence decisions), and the underlying rationale stay implicit. Practitioners do not capture decision knowledge because they fear intrusiveness and inconsistency, miss clear use cases for exploitation as well as techniques and tools. They see a potential benefit in supporting software evolution through captured non-existence decisions and decisions for code.

1) Types of Decisions not Captured: Seven practitioners provide examples for *executive decisions* regarding the CSE process that they do not capture but that they think would be important to capture. They state that the decisions on the continuous integration and deployment pipeline and the respective stages, e. g., the develop, test, and production stages, stay implicit in the head of developers. In total, eleven practitioners mention that they do not capture certain *existence decisions*. Such decisions relate to features, software

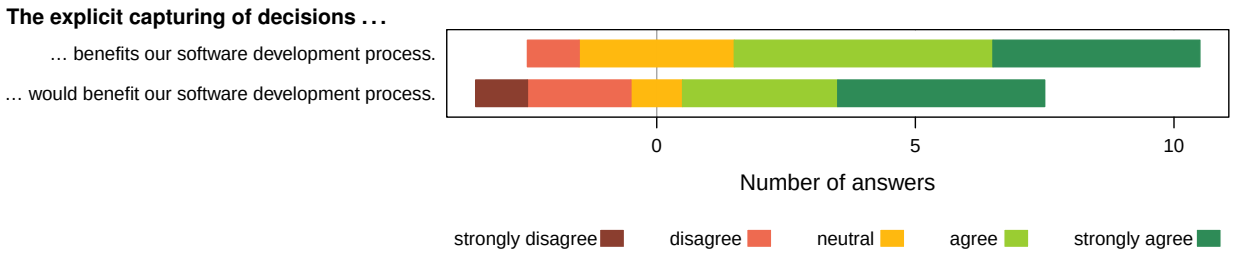

Figure 1. The practitioners' attitude towards capturing decisions (above) and towards capturing decisions that they currently do not capture (below).

architecture, implementation, and tests. For example, a practitioner reports to document application programming interfaces between microservices using *Swagger* but does not capture decisions for the design of such interfaces and the underlying rationale. The practitioners report to capture the outcome value regarding the prioritization of requirements based on cost estimation, test cases based on risk estimation, and bug fixing activities. However, the rationale is not captured, especially if it comes to reprioritization. Two practitioners report that they do not capture configuration decisions, e. g., which compiler or framework versions they use. Three practitioners criticize that they do not capture the rationale behind decisions and that they do not capture decisions on why they did not pick a certain alternative for an issue, i. e., *non-existence decisions or bans* stay implicit. Two practitioners mention that they have a common understanding of certain *property decisions*, e. g., about the coding style, but that such decisions are not documented. One practitioner provides the example that they did not capture the decision whether to use either a synchronous or an asynchronous inter-service communication between microservices. The practitioner states that these kind of decisions are made very quickly and then get reused by others, but are neither discussed nor captured.

2) Reasons why Decisions are not Captured: Two practitioners report that the decisions on how to deploy the software used to be captured in external documents but are no longer captured since the deployment is now automated. However, they still keep the former documents to externalize this knowledge. Four practitioners see a problem in *rapid changing decisions* that lead to outdated decisions, i. e., to inconsistency between the captured decisions and their implementation. One practitioner associates the waterfall process with capturing decision knowledge. Five practitioners report that they *lack appropriate techniques or tools* to capture decisions and rationale. Three of them state that their *process is not mature enough* to involve decision management. Six practitioners do not capture decisions because they *lack techniques for an easy retrieval and exploitation* of the captured decisions. Eight practitioners fear the *overhead and the intrusiveness* of capturing decisions and rationale. They could not spend the effort and do not have enough time. One practitioner mentions that the cost-benefit-ratio would be too high if they captured more decisions than they currently do according to the 80/20 rule. However, the practitioners admit that the extra effort could be reduced by applying better capturing techniques.

3) Potential Benefits if Captured: As for the captured decisions, practitioners see potential benefits in establishing accountability, improving decision-making and knowledge sharing, as well as a support of reuse and maintenance activities. They also stress that capturing decisions and rationale would support continuous learning as part of the CSE process. Two practitioners see a potential benefit in retrieving decisions and rationale for code when evolving code. In their opinion, this could *ease the understanding of code*. Three practitioners state that it would be useful for them to know about alternatives for a decision and the rationale why they were not selected during software evolution. One practitioner mentions disaster recovery as an example why knowledge sharing and capturing decisions was important.

We asked the practitioners to rate the statement *The explicit capturing of decisions would benefit our software development process* regarding the decisions that they currently do not capture. Practitioners of eleven interviews rated this statement: three disagreed, one was neutral, and seven agreed (Figure 1).

C. Sharing of Decision Knowledge during CSE

We dedicated two interview questions to address this research question.

> **RQ3**: *How do practitioners share decision knowledge?* Practitioners strongly rely on face-to-face communication, i. e., colleagues' knowledge, to recover implicit decisions. To share knowledge equally they apply techniques such as pair programming and inviting all team members as reviewers to pull requests. However, they also try to recover implicit decisions using reverse engineering.

1) Alternative Knowledge Sources: Six practitioners state that they try to do *reverse engineering* to recover knowledge from code and issue tracking systems. Ten practitioners mention that they *ask colleagues*, which has the disadvantage that both the inquiring person and the respondent need to interrupt their current activity. One practitioner reports that they have an *emergency mobile phone* that is carried by one knowledgeable project member for a period of time; afterwards, it is passed to the next project member. Two practitioners report that it can be hard to *scan through many emails and pull requests* to recover a decision. Thus, this decision was somehow documented but hard to retrieve. Another practitioner enforces that decisions are hard to retrieve in communication channels using the slogan "if it happens in [chat tool], it did not happen".

2) Avoidance of Knowledge Vaporization: The practitioners try to avoid knowledge vaporization by sharing knowledge between project members. One practitioner states that in larger teams it is both necessary to share the knowledge within and across team boundaries. Knowledge management should address both the intra- and inter-team scope. Within teams, the practitioners try to share knowledge between all members as homogeneously as possible. They strongly rely on face-to-face communication. Further, one practitioner mentions that they always *invite all team members as reviewers for pull requests* and also do *pair programming* to distribute knowledge. One practitioner states that they encourage team members to always share their notes with others, e. g., by using a wiki system, instead of "writing diaries". Two practitioners mention to have a dedicated process to onboard new project members. Generally, practitioners state that if a project member is about to leave the company, they would have a period of time during which this person tries to share and capture their knowledge.

D. Managing Changing Decisions during CSE

Overall, we received only few responses from practitioners regarding the management of changing decisions during CSE.

> **RQ4**: *How do practitioners deal with changing decisions?* Practitioners use cost and risk estimation as well as prioritization before integrating changing decisions. They depend on implicit knowledge and team communication to identify parts of the system affected by new or changed decisions. They rely on automated tests to detect side and ripple effects.

None of the practitioners report about a technique or tool to identify parts of the system that are affected by new or changed decisions. One practitioner reports about their *change management* process. For a change request, the project leader needs to decide whether the change will be integrated and—if so—the developers *estimate the cost* for the change, *define a priority*, and break it down into tasks. Other practitioners emphasize the importance of *automated tests* to detect side and ripple effects as well as *risk management*. One practitioner of a consulting company criticizes that workflows often do not scale when the project and the respective team sizes increase. Change impact analysis would be especially important for larger projects, however, it is not integrated since it had not been necessary at the beginning when the project was small.

IV. Discussion

In the following, we discuss the results in terms of findings, problems, and our improvement ideas.

From the results of our interview study, we cannot make a clear statement of which decisions are captured and which decisions stay implicit, since some practitioners mentioned to capture decisions that others do not capture and vice versa. Yet, the answers towards RQ1 and RQ2 provide examples of decisions practitioners consider important to be captured and for which purposes. It is interesting that many practitioners find executive decisions regarding the CSE process important to be captured. Reasons might be that CSE involves a continuous process improvement that comes with a continuous decision-making. The CSE process contains many defined workflows that developers need to decide on and for which they need to have a common understanding [1], [4].

Our findings confirm the challenges of CSE listed in Section I: In 19 interviews, the practitioners mention that their decision capturing method needs to be improved and that it is far from being perfect. Only in one interview, a practitioner in the role of a quality manager states that they are very focused to capture decisions. The degree of formalization of decision-making and documentation in practice seems to be rather low. During the interviews, only five practitioners mention to capture alternatives for a decision, i. e., non-existence decisions. However, Kruchten states that it is very important to capture non-existence decisions as they are not visible in the software artifacts and cannot be recovered using reverse engineering [2]. Also, the underlying rationale is not captured systematically. The practitioners argue to not capture decisions since the rapid change would make them outdated soon. Further, the practitioners state that the usage of too many tools for capturing decisions can be frustrating. As reasons they list a) redundancy, i. e., they need to document knowledge in more than one tool, which means twice the effort and might result in an inconsistent documentation, and b) a workflow interruption, i. e., they have to change their working context for documentation purposes, which means intrusiveness.

Although the practitioners confirm to document decision knowledge in typical documentation locations, e. g., the issue tracking system, the opportunities of CSE for an improved decision knowledge management are not yet exhausted. The practitioners stress that the utilization of the captured decision knowledge is not clear to them and that it is not exploited in a proper way. They also highlight that they have difficulties to find and retrieve the decisions—especially if captured in informal communication channels such as *Slack*. In summary, the capturing and exploitation of decision knowledge needs to be better integrated into the daily practices of developers.

We aim to provide solution proposals for these findings applicable for practitioners [12]. In [7] and [13], we describe ideas for a *continuous decision knowledge management (ConDec)* as part of CSE and in [14] we present a dashboard for knowledge visualization. The ConDec tool support[1] integrates with existing tools to minimize the intrusiveness of the decision knowledge management, e. g., with the issue tracking system. It provides many features for capturing and exploiting decision knowledge during CSE and supports knowledge sharing and changes. For example, ConDec enables the explicit, formal capture of decision knowledge in the description and comments of *JIRA* issues, commit messages, and code comments. It does not restrict the type of decisions, i. e., developers can capture executive, existence, non-existence, and property decisions. It enables developers to view decision knowledge in relation to software artifacts such as features and code. We evaluate the ConDec tool support in agile student courses [15].

[1] https://github.com/cures-hub

V. Threats to Validity

We conducted the interview study from a positivist philosophical stance, i. e., we try to draw conclusions on how practitioners manage decision knowledge during CSE from the interviews. We discuss the four criteria for validity as usually done for empirical research with a positivist stance [9], [8]. A more detailed discussion can be found in [4], [5].

Construct validity focuses on whether the theoretical constructs are measured and interpreted correctly. The practitioners might have interpreted the interview questions different to what we intended. To reveal misinterpretations, we allowed them to ask questions at any time and conducted two interviews with colleagues that we discussed afterwards. We used open-ended questions to elicit as much information as possible.

Internal validity concerns whether the results we draw really follow from the data, e. g., whether there are confounding factors that influence the results. The practitioners might have provided answers that do not fully reflect their daily work, since they knew that the results would be published. We guaranteed the full anonymity of interviewees and companies to address this. The interpretation of answers might be biased by the authors' *a priori* expectations, which we addressed by coding the transcriptions and discussing the codes.

External validity addresses the generalizability of the study results. We contacted companies that we already knew, which might result in a selection bias. It is mitigated by the fact that the authors are from two universities with different industrial contacts. 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. After we carried out coding training and checked intercoder reliability, two authors individually coded different transcripts. We addressed this threat by discussing questions during coding. In addition, a third author of this paper supervised the interview analysis.

VI. Related Work

Miesbauer and Weinreich collected 120 examples of decisions made in practice using an interview study [16]. Similar to our study, they classified these decisions according to Kruchten's taxonomy [2] and list documentation locations. They also found that the majority of the decisions were existence decisions and that property decisions were rarely mentioned. In contrast to our study, they found that practitioners did not mention non-existence decisions. They did not ask for practitioners' knowledge sharing and change management practices, but list influence factors for decision-making.

Similar to our work, Furtado *et al*. explore tools, processes, and benefits of knowledge management [17]. They found *Google Drive* and email lists applied most prevalent to capture knowledge and point out the value of the informal messaging service Slack to improve knowledge sharing. In contrast to our study, they did not focus on decision knowledge and their results are limited to one institution only.

VII. Conclusion

We reported on findings from an interview study on how practitioners manage decision knowledge in CSE environments. The practitioners mainly capture decision knowledge in an informal way, in wiki and issue tracking systems. The exploitation of the captured decision knowledge is partly unclear and needs to be improved. We develop techniques and tool support for a continuous decision knowledge management.

Acknowledgements

This work was supported by the DFG (German Research Foundation) under the Priority Programme SPP1593: Design For Future – Managed Software Evolution (CURES project). We thank the practitioners for sharing their insights.

References

[1] S. Krusche and B. Bruegge, "CSEPM - A continuous software engineering process metamodel," in *3rd International Workshop on Rapid Continuous Software Engineering*, 2017, pp. 2–8.

[2] P. Kruchten, "An ontology of architectural design decisions in software intensive systems," in *2nd Groningen Workshop on Software Variability Management*, 2004, pp. 54–61.

[3] A. H. Dutoit, R. McCall, I. Mistrík, and B. Paech, *Rationale Management in Software Engineering*. Springer, 2006.

[4] J. O. Johanssen, A. Kleebaum, B. Paech, and B. Bruegge, "Practitioners' eye on continuous software engineering: An interview study," in *Int. Conf. on Softw. and System Process (ICSSP)*. ACM, 2018, pp. 41–50.

[5] ——, "Continuous software engineering and its support by usage and decision knowledge: An interview study with practitioners," *Journal of Software: Evolution and Process*, 2019.

[6] J. O. Johanssen, A. Kleebaum, B. Bruegge, and B. Paech, "Towards a systematic approach to integrate usage and decision knowledge in continuous software engineering," in *2nd Workshop on Continuous Software Engineering*, 2017, pp. 7–11.

[7] A. Kleebaum, J. O. Johanssen, B. Paech, and B. Bruegge, "Tool support for decision and usage knowledge in continuous software engineering," in *3rd Workshop on Cont. Softw. Eng.*, 2018, pp. 74–77.

[8] P. Runeson, M. Host, A. Rainer, and B. Regnell, *Case Study Research in Softw. Eng.: Guidelines and Examples*. John Wiley & Sons, 2012.

[9] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian, "Selecting empirical methods for softw. engineering research," in *Guide to Advanced Empirical Softw. Eng.* London: Springer, 2008, ch. 11, pp. 285–311.

[10] K.-J. Stol and B. Fitzgerald, "The ABC of Software Engineering Research," *ACM Transactions on Software Engineering and Methodology*, vol. 27, no. 3, pp. 1–51, 2018.

[11] J. Saldaña, *The Coding Manual for Qualitative Researchers*, 2nd ed. SAGE Publications, 2009.

[12] A. S. Freire, A. Meireles, G. Guimarães, M. Perkusich, R. M. da Silva, K. C. Gorgônio, A. Perkusich, and H. O. Almeida, "Investigating gaps on agile improvement solutions and their successful adoption in industry projects - A systematic literature review," in *30th Int. Conf. on Software Engineering and Knowledge Engineering*, 2018, pp. 40–45.

[13] A. Kleebaum, J. O. Johanssen, B. Paech, R. Alkadhi, and B. Bruegge, "Decision knowledge triggers in continuous software engineering," in *4th Int. Workshop on Rapid Cont. Softw. Eng.*, 2018, pp. 23–26.

[14] J. O. Johanssen, A. Kleebaum, B. Bruegge, and B. Paech, "Towards the Visualization of Usage and Decision Knowledge in Continuous Software Engineering," in *2017 IEEE Working Conference on Software Visualization*, vol. 1806. IEEE, sep 2017, pp. 104–108.

[15] A. Kleebaum, J. O. Johanssen, B. Paech, and B. Bruegge, "Teaching rationale management in agile project courses," in *16. Workshop Softw. Eng. im Unterricht der Hochschulen (SEUH)*, 2019, pp. 125–132.

[16] C. Miesbauer and R. Weinreich, "Classification of design decisions - an expert survey in practice," in *7th European Conference on Software Architecture (ECSA'13)*, K. Drira, Ed. Springer, 2013, pp. 130–145.

[17] F. S. Furtado, G. Alexandre, N. G. de Sá Leitão Júnior, I. H. de Farias Junior, and H. P. Moura, "Knowledge management in a software development organization: Identifying tools, processes and benefits," in *29th Int. Conf. on Softw. Eng. and Knowl. Eng.*, 2017, p. 627.