Towards a Systematic Approach to Integrate Usage and Decision Knowledge in Continuous Software Engineering

Jan Ole Johanssen[1] Anja Kleebaum[2] Bernd Bruegge[3] Barbara Paech[4]

Abstract: Continuous software engineering (CSE) employs activities such as continuous integration and continuous delivery to support software evolution. Another aspect of software evolution is knowledge management. There are two important knowledge types: usage knowledge derives from explicit and implicit user feedback and helps to understand how users utilize software. Decision knowledge encompasses decisions and their rationale on all aspects of the software lifecycle. Both knowledge types represent important information sources for developers to improve the CSE activities and the software product. We envision an integration of usage and decision knowledge in the CSE lifecycle. This extension consists of a monitoring and feedback component for user understanding as well as a knowledge repository and dashboard component for knowledge visualization and analysis. Usage and decision knowledge introduce challenges when integrating them in CSE. In this paper, we present our vision and detail the challenges.

Keywords: Continuous Software Engineering, Usage Knowledge, Decision Knowledge

1 Introduction

Continuous software engineering (CSE) has been described by Bosch [Bo14] as a holistic approach for software engineering that consists of several activities covering the complete software lifecycle including continuous integration and continuous delivery.

Fitzgerald and Stol indicate the importance of monitoring any behavior during the run-time of a system in CSE [FS15]. We refer to insights derived from users either through testing or normal use as *usage knowledge*. Decisions made during the CSE activities represent another type of knowledge: *decision knowledge*. Creating and managing decision knowledge has already been explored by several researchers such as Dutoit et al. [Du06], while they focused on design-time knowledge only. However, with its focus on evolution during run-time, CSE introduces new challenges on managing usage and decision knowledge.

We have already worked on the integration of both usage and decision knowledge in non-CSE environments: for instance, regarding the analysis of user behavior, we compared monitored user interaction to use cases [Ro13]. Furthermore, regarding decision management, we developed the decision documentation model as well as corresponding tool support and explored its application [HKR16].

[1] Technische Universität München, Munich, Germany, johansse@in.tum.de
[2] Universität Heidelberg, Heidelberg, Germany, anja.kleebaum@informatik.uni-heidelberg.de
[3] Technische Universität München, Munich, Germany, bruegge@in.tum.de
[4] Universität Heidelberg, Heidelberg, Germany, paech@informatik.uni-heidelberg.de

Up to now, usage and decision knowledge have not been explored within CSE. In our research project CURES (”Continuous Usage- and Rationale-based Evolution Decision Support”), we plan to integrate both knowledge types in CSE. Beside their individual benefits, we expect to leverage synergies in their combination: for instance, decisions might rely on results derived from detected usage patterns. In addition, both face similar questions regarding a consistent way of defining and storing the right granularity of information.

In the following, we present our vision of integrating usage and decision knowledge in CSE. Thereafter, we outline integration challenges and describe our status and next steps.

2 Vision

As shown in Fig. 1, we envision an extension to CSE in which developers and users act as the main stakeholders. As for the CSE infrastructure, we plan to employ Rugby, an agile process model for continuous delivery which builds upon event-based releases [Kr14]. Developers utilize feature branches to add product increments in form of code commits. Feature branches are merged from and back to a master branch, which contains the final software product. Each feature branch on its own can be released to users, allowing the delivery of different proposals for one feature at the same time. Using a monitoring and feedback component, developers can examine usage information that is mapped to individual releases by applying approaches such as A/B testing for implicit usage information.

A knowledge repository continuously stores all information related to the development and monitoring process. Furthermore, it maintains decisions related to feature branches. Thereby, rationale can be accessed, visualized, and analyzed using a dashboard component. For instance, the claimed solution to an issue could be compared to an alternative solution based on the impact of a feature branch in the context of previous decisions. This enables the developer to interact and reflect on collected usage and decision knowledge.

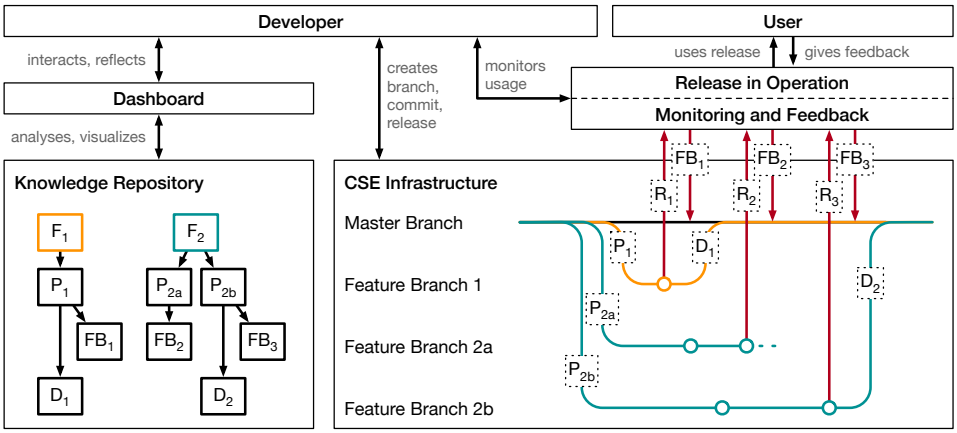

Fig. 1: Interaction between CSE Infrastructure, Knowledge Repository, and Dashboard: Feature *F* is based on a Proposal *P*, Feedback *FB* is collected from a Release *R* and leads to a Decision *D*.

For example, a feature F_1 is developed based on a proposal P_1. After providing the release R_1 to users and analyzing feedback FB_1, the developer makes decision D_1 and merges F_1 into the master branch. In the same manner, different proposals P_{2a} and P_{2b} for a feature F_2 are evaluated. Feedback FB_2 stops the work on P_{2a}, while FB_3 leads to decision D_2.

This CSE infrastructure extension forms a *continuous knowledge* activity and enables the exploitation and documentation of knowledge acquired during CSE. It supports the evolution of knowledge and handles it similar to the evolution of code in a CSE infrastructure.

3 Challenges

In this section we describe the challenges of integrating usage and decision knowledge in CSE and highlight references from the extension introduced in Fig. 1 in bold type.

There is a heterogeneous set of users. In CSE, the **user** role might be filled by the developers themselves, the customers who initiate the development of an application as part of a contract, or the actual end users of an application. All of them have different intentions, expectations, and approaches towards using an application which needs to be considered.

User **feedback** can either be explicit or implicit. Explicit feedback is actively provided by the user, such as an app store review, while implicit feedback relates to anything users reveal through their interaction, such as pressing a button. It has yet to be answered how both feedback types can be incorporated within the CSE infrastructure: different strategies for feedback elicitation need to be evaluated, such as in-application feedback requests for new features or automatic usage data collection for each feature branch.

Enriching users' explicit with implicit feedback in a **monitoring and feedback** component enables the creation of a comprehensive user understanding. A main challenge is to derive a user's motivation and construct a user behavior model. Additional context information, such as a user's location or current activity, might be useful for creating such a model.

Further challenges arise in the synchronization of releases, feedback, and their granularity: in a **CSE infrastructure** new increments are released frequently, while usage patterns develop only over time and the difference between releases can be as little as a commit.

The decision documentation model is our starting point to integrate decision knowledge in CSE since it allows for an intertwined documentation across agile development activities. We want to address the following challenges towards a continuous rationale management.

Due to CSE's focus on run-time, decision making is a high-frequency process and leads to large amounts of data. **Developers** utilize different components, such as issue tracking and version control systems. Knowledge is distributed across these components: for instance, requirements are stored in the issue tracking system, whereas source code is stored in the version control system. It is a challenge how distributed decision knowledge can be systematically managed and synchronized during CSE as part of a **knowledge repository**.

Regarding the decision knowledge repository further questions arise: (1) Which decisions are worthwhile to be captured and managed? Should they be captured within the feature branch or the master branch? (2) What is the right granularity of decisions and the related rationale? (3) How to enable a seamless switch between coarse and fine-grained decision knowledge? (4) What is the most suitable way to search for decision knowledge related to a problem during software evolution? How should the knowledge be presented in the **dashboard**? (5) How can a change impact analysis on decision knowledge be performed efficiently and effectively?

4 Status and Next Steps

Currently, we work on an empirical study to investigate on CSE needs in practice. The goal is to determine how developers deal with usage and decision knowledge to align the results with our next steps. We develop a decision documentation editor for JIRA and Git. This is our initial step to integrate the decision documentation model in the CSE infrastructure. We plan on applying machine learning techniques to classify implicit user feedback.

5 Conclusion

Usage and decision knowledge has not been explored in CSE up to now. We described our vision for a systematic approach to integrate both aspects in CSE. We identified challenges such as the heterogeneity of users and differences in explicit and implicit feedback. Another challenge is the alignment of decision making processes with respect to the fast pace of CSE activities such as continuous delivery, resulting in questions on decision capturing and granularity. Finally, we outlined our status and next steps for the integration of usage and decision knowledge in order to improve existing CSE infrastructures.

Acknowledgement

This work was supported by the DFG (German Research Foundation) under the Priority Programme SPP1593: Design For Future – Managed Software Evolution.

References

[Bo14] Bosch, Jan: Continuous Software Engineering: An Introduction. Springer, 2014.

[Du06] Dutoit, Allen H; McCall, Raymond; Mistrík, Ivan; Paech, Barbara: Rationale Management in Software Engineering: Concepts and Techniques. Springer, 2006.

[FS15] Fitzgerald, Brian; Stol, Klaas-Jan: Continuous software engineering: A roadmap and agenda. Journal of Systems and Software, 2015.

[HKR16] Hesse, Tom-Michael; Kuehlwein, Arthur; Roehm, Tobias: DecDoc: A Tool for Documenting Design Decisions Collaboratively and Incrementally. In: Proceedings of the 1st Int. Workshop on Decision Making in Software ARCHitecture. pp. 30–37, 2016.

[Kr14] 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. pp. 42–50, 2014.

[Ro13] Roehm, Tobias; Bruegge, Bernd; Hesse, Tom-Michael; Paech, Barbara: Towards Identification of Software Improvements and Specification Updates by Comparing Monitored and Specified End-User Behavior. In: Proceedings of the 29th IEEE International Conference on Software Maintenance. pp. 464–467, 2013.