Continuous Management of Requirement Decisions Using the ConDec Tools

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

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

Abstract

[Context and motivation] While eliciting, prioritizing, and implementing requirements, requirements engineers and developers continuously make decisions. They establish important decision knowledge that needs to be documented and exploited, i. e., thoroughly managed, so that it contributes to the evolution and future changes of a software system. **[Question/problem]** The management of decision knowledge is difficult for various reasons: 1) The documentation process is an additional effort, i. e., it is intrusive in the development process. 2) The documented knowledge can be of low quality in terms of completeness and consistency. 3) It might be distributed across many documentation locations, such as issue comments and commit messages, and thus difficult to access and use. **[Principal ideas/results]** Continuous software engineering (CSE) emerged as a development process that involves frequent, incremental decision making, implementation, and validation of requirements. During CSE, requirements engineers and developers implicitly document decision knowledge during established practices, such as committing code, working with issues in an issue tracking system, or conducting meetings. That means that CSE offers opportunities for the non-intrusive capturing of decision knowledge in various documentation locations. **[Contribution]** We develop the ConDec tools (https://se.ifi.uni-heidelberg.de/condec.html) that support requirements engineers and developers in documenting and exploiting decision knowledge directly within the tools they use, such as issue tracking and wiki systems, related to various software artifacts, such as requirements and code, and during change impact analysis.

1 Introduction

Continuous software engineering (CSE) is an agile software development process that supports the incremental implementation of requirements and their timely validation through user feedback [9]. During CSE, requirements engineers and developers make and realize decisions in a frequent and iterative manner. In particular, they establish important decision knowledge, i. e., knowledge about decisions and their rationale related to the requirements

In: M. Sabetzadeh, A. Vogelsang, S. Abualhaija, M. Borg, F. Dalpiaz, M. Daneva, N. Fernández, X. Franch, D. Fucci, V. Gervasi, E. Groen, R. Guizzardi, A. Herrmann, J. Horkoff, L. Mich, A. Perini, A. Susi (eds.): Joint Proceedings of REFSQ-2020 Workshops, Doctoral Symposium, Live Studies Track, and Poster Track, Pisa, Italy, 24-03-2020, published at http://ceur-ws.org

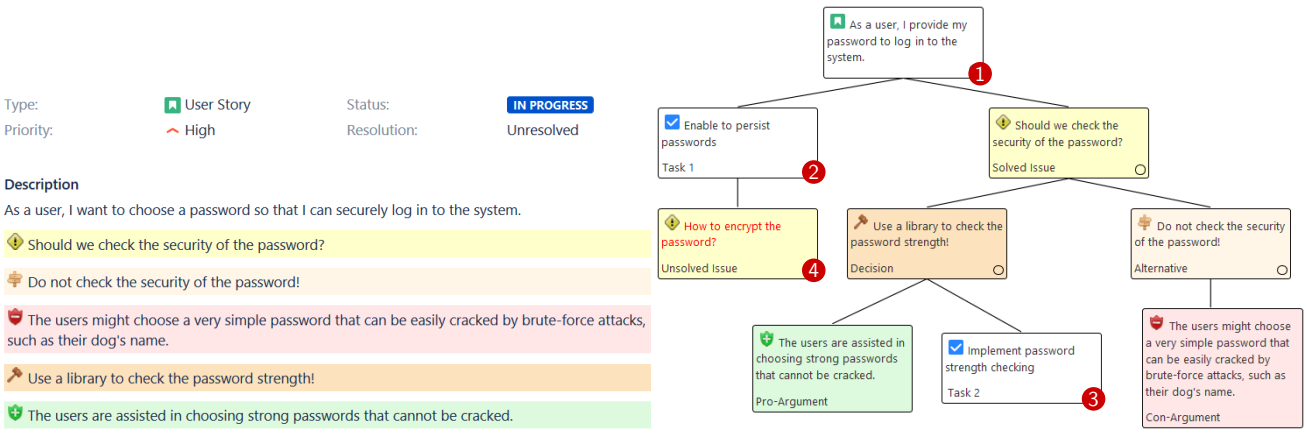

Figure 1: **Left**: Decision knowledge captured in the description of a Jira user story. **Right**: The user story ①, development tasks ②, ③, and decision knowledge visualized as a tree. The summary of the unsolved issue ④ is highlighted in red font. The knowledge tree is interactive, e. g., via a context menu and drag & drop possibility.

of the software under development. It has been known for long that the management of decision knowledge is important to evolve a software system and to cope with future changes [4]. For example, Roeller et al. [19] state that "decisions are like material floating in a pond. When not touched for a while, they sink and disappear from sight [...] [and] are the most difficult to change at a later stage." The management of decision knowledge comprises its documentation and exploitation. It is also referred to as rationale management [8].

Several reasons hinder requirements engineers and developers from systematically managing decision knowledge. Problems are 1) the intrusiveness and overhead of the documentation in a separate tool, 2) a low decision knowledge documentation quality in terms of incompleteness and obsolescence, i. e., inconsistency, and 3) the distributed documentation locations, which make the documented decision knowledge hard to exploit [12]. CSE and current software development tools provide opportunities to minimize the overhead of the documentation of decision knowledge. Issue tracking and version control systems offer lightweight documentation locations, such as issue comments, commit messages, and code comments used in established practices such as committing code. However, challenges remain in the low quality of the decision knowledge documentation and the exploitation of knowledge in various documentation locations. Our long term vision is an on-demand decision documentation as part of the on-demand developer documentation [18] in that requirements engineers and developers continuously document, share, and exploit decision knowledge. In our previous work [11, 14], we presented requirements for the management of decision knowledge in CSE. In this paper, we describe the resulting ConDec tools for the continuous management of decision knowledge that aim to tackle the three problems of rationale management. To overcome the intrusiveness, ConDec directly integrates into the development tools that requirements engineers and developers often use, for example, into the issue tracking system, version control system, wiki system, chat system, and the integrated development environment. Currently, the ConDec tools consist of a Jira, an Eclipse, a Confluence, a Slack, and a Bitbucket plugin. The ConDec tools are open source.[1]

The remainder of this paper is structured as follows: Section 2 presents the ConDec features and views, whereas Section 3 summarizes important design decisions. Section 4 sketches a scenario on the tools' usage. In Section 5, we present evaluation plans. Section 6 discusses related work and Section 7 concludes the paper.

2 ConDec Features and Views

In the following, we present an overview of the features (*italic highlighting*) and views of the ConDec tools.

Knowledge consists of knowledge elements and relationships among them. Knowledge elements are software artifacts such as requirements, development tasks, source code, and also decision knowledge elements. ConDec enables to *configure* the model that defines which decision knowledge elements and relationships are used. The default decision knowledge elements are: the issue, i. e. decision problem, to be solved, alternatives, pro- and con-arguments, and the decision. ConDec supports bidirectional relationships that are commonly used to model the interdependencies between solution options, i. e., between decisions and alternatives [15]: enables, constrains, forbids, comprises, subsumes, overrides, and relates. In addition, pro-arguments support solution

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

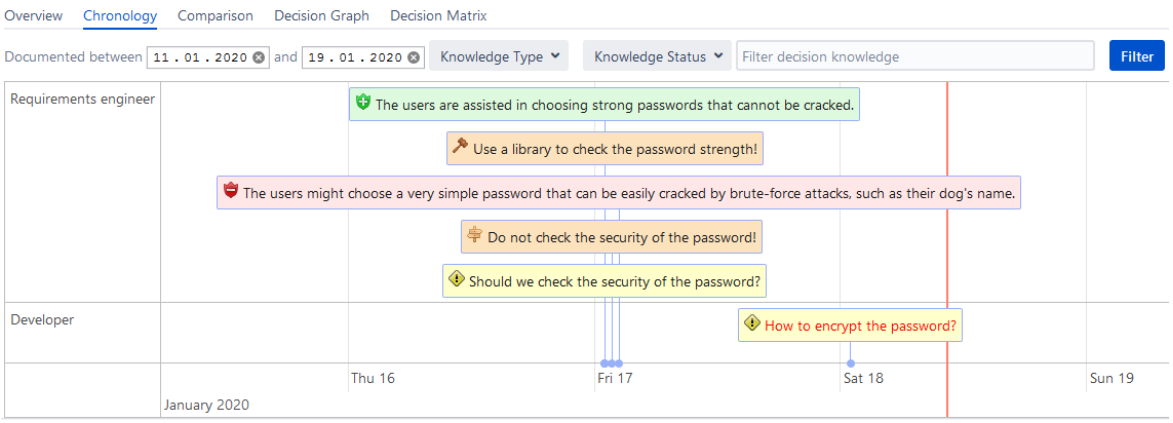

Figure 2: Chronology view with filters. The summary of the unsolved issue is highlighted in red font.

options, whereas con-arguments attack solution options. ConDec enables requirements engineers and developers in *documenting decision knowledge* directly in the tools that they work in. In Jira, they document decision knowledge within the description and comments of existing Jira issues such as user stories or tasks (Figure 1, left). Also, they can create new Jira issues with distinct types of decision knowledge. Besides, ConDec supports the documentation of decision knowledge in code comments, commit messages, Confluence wiki pages, and Slack chat messages. Requirements engineers and developers manually mark text as a respective decision knowledge element. They are supported by a *text classifier* that automatically identifies decision knowledge elements.

ConDec builds up and *visualizes the knowledge graph* consisting of knowledge elements. The knowledge graph is accessible from single knowledge elements such as requirements and code classes. The right side in Figure 1 shows a view included in a Jira issue. ConDec provides other views, such as an overview of all decision knowledge elements in a project, a view for the entire knowledge graph, a chronology view as suggested by Lee and Kruchten [16] (Figure 2), and a matrix view as suggested by Boer et al. [2]. Requirements engineers and developers use the integrated knowledge visualization to understand the requirements of a software system along with decisions related to their elicitation or implementation. They make new decisions consistent with the requirements and former decisions by exploiting the knowledge graph during changes, to *estimate change impacts*. ConDec offers various *filters*. Filter criteria are the textual content of knowledge elements, the types of knowledge elements and relationships, status, documentation location, distance in the knowledge graph, and time. For example, requirements engineers could view decisions for a requirement only, without seeing the tasks and alternatives.

Requirements engineers are supported in creating *meeting agendas* filled with decision knowledge matching the given filter criteria (Figure 3, left) and in documenting decision knowledge in meetings. The ConDec Slack tool comprises a *chat bot* that supports both the requirements engineers and the developers in *exporting decision knowledge captured in chat channels* to the respective requirement. Besides, the chat bot informs them about new decision problems and decisions regarding requirements in an *information channel*. The communication is supported through *automated release notes* including decision knowledge for the particular release. To exploit the documentation, it must be of high quality. ConDec supports *quality checking* using dashboards and *enforcement* of the complete documentation of decision knowledge. The completeness can be a quality gate in pull requests so that they can only be accepted and the branch be merged if the quality check is passed (Figure 3, right).

3 ConDec Architecture

This section presents decisions concerning the ConDec architecture. The ConDec tools are plugins for existing development tools and thus bound to the respective plugin framework and programming languages. Mainly, the tools are written in Java, JavaScript, and other languages for web development. An issue was where to persist and how to synchronize the knowledge from different locations. We decided to use Jira and git as the central decision knowledge repositories. Decision knowledge from other locations such as chat messages is exported to Jira and then maintained there—close to the requirements and code. To enable the explicit capturing of decision knowledge in the description and comments of Jira issues and their linking with other knowledge elements, the ConDec Jira plugin introduces two new database tables created with the active objects framework. The knowledge graph is represented with the jGraphT library similarly to [5]. It is a singleton object and is only updated by changes, e. g., when a new element is added. The alternative would be to recreate the entire knowledge

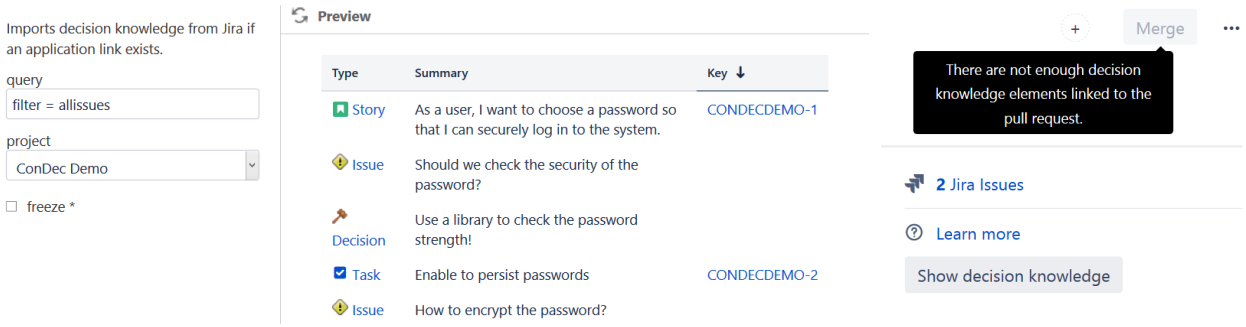

Figure 3: **Left**: Macro to create a meeting agenda as part of the ConDec Confluence plugin. **Right**: Merge check for decision knowledge completeness as part of the ConDec Bitbucket plugin.

Figure 4: **Left**: Commit message containing a design decision. The developer works on a branch related to a development task. **Right**: Subtree of the knowledge tree in Figure 1 with the design decision as a new leaf node. The formerly open issue is now solved.

graph for every user interaction, which is not efficient. For graph visualization, ConDec uses the jstree, treant, and vis.js JavaScript libraries and the gephi library in Eclipse. REST APIs are extensively used throughout all the plugins for communication. The ConDec Eclipse plugin uses the Jira REST Java client library to access Jira's REST API. The ConDec Jira plugin offers a REST API to document and access decision knowledge. For example, this REST API is used in the ConDec Slack plugin to export decision knowledge elements from Slack to Jira. Git repositories, e. g., commits related to Jira issues, are accessed by using the jgit library.

4 ConDec Scenario

In the following, we describe a scenario to demonstrate how ConDec supports knowledge management for requirements engineers and developers. In Figure 1, the requirements engineer creates the user story *As a user, I want to choose a password so that I can securely log in to the system* and captures the issue whether passwords should pass a security check as well as the decision to integrate a library for it. The requirements engineer creates the following development tasks that split the requirement: *Enable to persist passwords* and *Implement password strength checking*. The developer assigned to the task captures the issue *How to encrypt the password?* in a comment of the first task. ConDec visualizes the knowledge graph and automatically appends this issue to the knowledge tree (Figure 1-④). During a meeting, the requirements engineer and developers spot unsolved decision problems from the meeting agenda (Figure 3, left). They orally decide to encrypt passwords with the SHA1 algorithm but do **not** document it. The developer implements the first task on a feature branch. The branch cannot be merged back to the master branch as long the issue is still unsolved (Figure 3, right). The developer makes a commit on this branch with the decision *Encrypt the password with the SHA1 algorithm* being part of the commit message (Figure 4, left). The text classifier automatically identifies the decision in the commit message. Since the branch and the commits are linked to the task, ConDec automatically relates the decision to the issue. The issue is marked as solved and the red highlighting is removed (Figure 4, right). The developer is allowed to merge the branch back to the mainline as the decision knowledge documentation is complete in the sense that both the issue and the decision are documented.

5 Evaluation

For the evaluation of the ConDec tools, we use the following design that targets agile project courses with students. We introduce students to rationale management by giving an introductory lecture at the beginning of a course [13]. Afterward, the students apply the ConDec tools over the duration of one semester. The evaluation

addresses two goals: The first goal is to show the acceptance of the ConDec tools. The students participating in the evaluation need to answer questions in a written form and during interviews. We derive the questions in a questionnaire by considering the variables of the technology acceptance model [7]: the perceived usefulness, the perceived ease of use, and the intention to use. We ask the students to provide detailed feedback on the features for the management of decision knowledge they applied. In particular, we want to evaluate the text classification feature. The accuracy of the text classifier is not perfect and differs for every project and developer. We need to evaluate to what extent the developers are triggered to explicitly capture decision knowledge even if the text classifier has false positives. The second goal is to assess the quality of the established data sets in terms of consistency and completeness of the documented knowledge.

Using the design described above, we already collected data from two teams of a multi-project course at the Technical University of Munich. Currently, we are gathering additional data from another course at Heidelberg University. The projects at university aim to simulate an industrial environment and do have industrial customers. Nevertheless, it would be beneficial to evaluate the ConDec tools in real industry since practitioners might have a better feeling of decision-making issues in industrial projects than students. Currently, we have no plans for implementing the ConDec tools in an industrial environment outside the university. However, we are positive that the ConDec tools could be installed in industrial projects as long as the underlying tools fit, e. g., Jira is used as the issue tracking system.

6 Related Work

This section discusses related tools for the management of decision knowledge. Hesse et al. [10] and Capilla et al. [4] list and compare existing tools, such as SEURAT [3], the Decision Architect [17], and Archie [6]. Many of these existing tools focus on the documentation of architectural design decisions as they are difficult to change in the future and thus very important. ConDec enables to capture architectural design decisions but also other kinds of decisions, e. g., related to the elicitation and prioritization of requirements.

The first goal of ConDec is to *minimize the intrusiveness of the documentation and exploitation* of decision knowledge. For this, ConDec enables requirements engineers and developers to document decision knowledge in various locations, such as chat messages, commit messages, and issue comments, which is different to the tools mentioned above. Besides, ConDec enables the requirements engineers and developers to access the documentation from within the various tools they work with so that they do not have to change their development context. To support easy documentation, ConDec offers a text classification feature. Bhat et al. [1] develop a tool to manage architectural design decisions that also offers a text classification feature for decision knowledge. Their classifier identifies different types of decisions, e. g., existence and ban decisions [15], whereas our classifier identifies different types of decision knowledge, i. e., the issue (decision problem), alternatives, the decision, as well as pro- and con-arguments. Our classifier does not distinguish different types of decisions but we consider this as a future improvement. Bhat et al. [1] use the architecture knowledge management system AMELIE as the central repository and also the viewpoint for software artifacts such as requirements, stakeholders, and architectural elements, and decisions. ConDec uses Jira and git as the central repositories for decision knowledge, requirements, and code. In ConDec, the decision knowledge views are part of many tools.

The second goal of ConDec is to *maximize the quality of the documented decision knowledge* in terms of completeness and consistency. SEURAT also offers metrics to infer whether the rationale documentation is complete and produces warnings if it detects incompleteness, e. g., if an alternative has more pro-arguments than the decision [3]. ConDec uses similar metrics. Besides, it offers the merge check in pull requests and dashboards.

7 Conclusion and Future Work

ConDec supports requirements engineers and developers in managing decision knowledge in a lightweight and non-intrusive way. It integrates knowledge from different documentation locations and provides a means to support and assess the documentation quality. In our future work, we will evaluate on how the decision knowledge helps new team members in getting insight into the system's decision history, improves the ongoing development, and supports future changes. We plan to improve the support for change impact analysis. Currently, ConDec supports manual inspection of the relationships between decisions and other knowledge elements in the knowledge graph, which can be used to investigate how a change in a decision may affect other decisions. Carrillo and Capilla [5] describe a more advanced ripple effect algorithm and a technique to assess the stability of decisions. For this, they also categorize dependency types between decisions into only four categories. We will adopt this simplification, but also enable the ConDec users to configure more dependency types.

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 students who contribute to the ConDec tools as well as Doris Keidel-Mueller and the anonymous reviewers.

References

[1] M. Bhat, C. Tinnes, K. Shumaiev, U. Hohenstein, A. Biesdorf, and F. Matthes. "A-DEX: A Tool For Automatic Curation of Design Decision Knowledge and Recommendation of Alternatives and Experts". In: *International Conference on Software Architecture (ICSA 2019)*. Hamburg, Germany, 2019.

[2] R. C. de Boer, P. Lago, A. Telea, and H. V. Vliet. "Ontology-Driven Visualization of Architectural Design Decisions". In: *Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture*. Cambridge, UK: IEEE, 2009, pp. 51–60.

[3] J. E. Burge and D. C. Brown. "Software Engineering Using RATionale". In: *Journal of Systems and Software* 81.3 (2008), pp. 395–413.

[4] R. Capilla, A. Jansen, A. Tang, P. Avgeriou, and M. A. Babar. "10 years of software architecture knowledge management: Practice and future". In: *Journal of Systems and Software* 116 (2016), pp. 191–205.

[5] C. Carrillo and R. Capilla. "Ripple effect to evaluate the impact of changes in architectural design decisions". In: *12th European Conf. on Software Architecture (ECSA'18)*. Madrid, Spain: ACM Press, 2018, pp. 1–8.

[6] J. Cleland-Huang, M. Mirakhorli, A. Czauderna, and M. Wieloch. "Decision-Centric Traceability of architectural concerns". In: *7th International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*. San Francisco, CA, USA: IEEE, 2013, pp. 5–11.

[7] F. D. Davis, R. P. Bagozzi, and P. R. Warshaw. "User Acceptance of Computer Technology: A Comparison of Two Theoretical Models". In: *Mangagement Science* 35.8 (1989), pp. 982–1002.

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

[9] B. Fitzgerald and K.-J. Stol. "Continuous software engineering: A roadmap and agenda". In: *Journal of Systems and Software* 123 (2017), pp. 176–189.

[10] T.-M. Hesse, A. Kuehlwein, and T. Roehm. "DecDoc: A Tool for Documenting Design Decisions Collaboratively and Incrementally". In: *1st International Workshop on Decision Making in Software ARCHitecture (MARCH 2016)*. Venice, Italy: IEEE, 2016, pp. 30–37.

[11] 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 Continuous Softw. Engineering*. 2018, pp. 74–77.

[12] A. Kleebaum, J. O. Johanssen, B. Paech, and B. Bruegge. "How do Practitioners Manage Decision Knowledge during Continuous Software Engineering?" In: *31st International Conference on Software Engineering and Knowledge Engineering*. SEKE'19. Lisbon, Portugal: KSI Research Inc., 2019, pp. 735–740.

[13] A. Kleebaum, J. O. Johanssen, B. Paech, and B. Bruegge. "Teaching Rationale Management in Agile Project Courses". In: *16. Workshop SEUH*. Bremerhaven, Germany, 2019, pp. 125–132.

[14] A. Kleebaum, M. Konersmann, M. Langhammer, B. Paech, M. Goedicke, and R. Reussner. "Continuous Design Decision Support". In: *Managed Software Evolution*. Cham: Springer, 2019. Chap. 6, pp. 107–139.

[15] P. Kruchten. "Documentation of Software Architecture from a Knowledge Management Perspective – Design Representation". In: *Software Architecture Knowledge Management*. Springer, 2009, pp. 39–57.

[16] L. Lee and P. Kruchten. "A Tool to Visualize Architectural Design Decisions". In: *Quality of Software Architecture Models and Architecture*. Springer, 2008, pp. 43–54.

[17] C. Manteuffel, D. Tofan, P. Avgeriou, H. Koziolek, and T. Goldschmidt. "Decision architect - A decision documentation tool for industry". In: *Journal of Systems and Software* 112 (2015), pp. 181–198.

[18] M. P. Robillard, A. Marcus, C. Treude, G. Bavota, O. Chaparro, N. Ernst, M. A. Gerosa, M. Godfrey, M. Lanza, M. Linares-Vásquez, G. C. Murphy, L. Moreno, D. Shepherd, and E. Wong. "On-Demand Developer Documentation". In: *International Conference on Software Maintenance and Evolution*. 2017, p. 5.

[19] R. Roeller, P. Lago, and H. van Vliet. "Recovering architectural assumptions". In: *Journal of Systems and Software* 79.4 (2006), pp. 552–573.