Sharing and Exploiting Requirement Decisions

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

1 Introduction

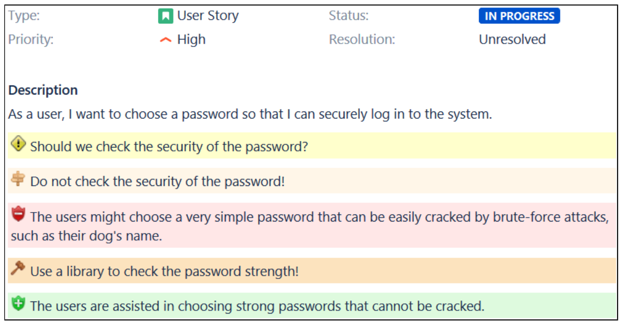
Continuous software engineering is an agile development process that puts particular emphasis on the incremental implementation of requirements and their rapid validation through user feedback. This involves frequent and incremental decision making, which needs to be shared within the team. Requirements engineers and developers have to share their decision knowledge since the decisions made are particularly important for future requirements. It has been a vision for long that important decision knowledge gets documented and shared [1]. However, several reasons hinder requirements engineers and developers from doing this, for example, the intrusiveness and overhead of the documentation [2]. Current software development tools provide opportunities to minimize the overhead. Issue tracking and version control systems offer lightweight documentation locations, such as issue comments, commit messages, and code comments. With ConDec, we develop tool support for the continuous management of decision knowledge that uses techniques for natural language processing and integrates into tools that developers often use[1], for example, into the issue tracking system Jira[2]. In this work, we focus on how ConDec enables requirements engineers and developers to share and exploit decision knowledge regarding requirements.

2 ConDec Usage Scenario

We describe a scenario to demonstrate how ConDec improves knowledge sharing between requirements engineers and developers. In our example, the requirements engineer creates the following requirement in the form of a user story: *As a user, I want to choose a password so that I can securely log in to the system.*

The requirements engineer captures decision knowledge directly in the description of the requirement. Decision knowledge consists of elements of various types: the issue, i. e. decision problem, to be solved, options for the solution, pro- and con-arguments, and the solution decision. The elements are linked and build up a knowledge graph. In Figure 1, they capture the issue whether passwords should pass a security check and the decision to integrate a library for it. They can manually mark text as decision knowledge but are also supported by a text classifier that automatically identifies decision knowledge elements. ConDec visualizes the relationships between knowledge elements as a knowledge graph or tree with the requirement being the root (Figure 2-①).

The requirements engineer creates development tasks that split the requirement: *Enable to persist passwords* (Figure 2-②) and *Implement password strength checking* (Figure 2-③). They capture the issue *How to encrypt the password?* in a comment of the first task. ConDec automatically appends this issue to the knowledge tree (Figure 2-④).


Type:	User Story	Status:	IN PROGRESS
Priority:	High	Resolution:	Unresolved

Description
As a user, I want to choose a password so that I can securely log in to the system.

Should we check the security of the password?

Do not check the security of the password!

The users might choose a very simple password that can be easily cracked by brute-force attacks, such as their dog's name.

Use a library to check the password strength!

The users are assisted in choosing strong passwords that cannot be cracked.

Figure 1: Decision knowledge captured in the description of the user story in Jira.

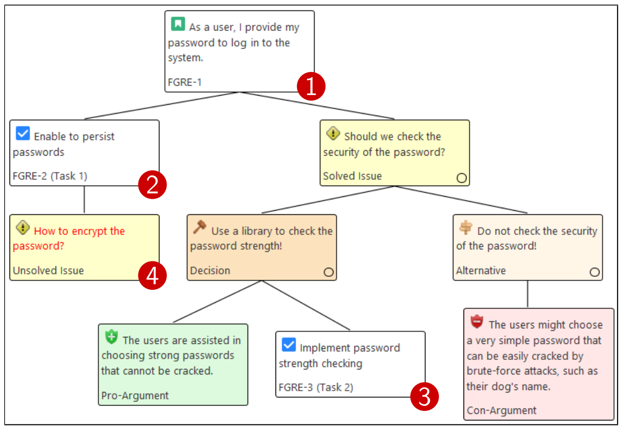

As a user, I provide my password to log in to the system.
FGRE-1
1
Enable to persist passwords
FGRE-2 (Task 1)
2
Should we check the security of the password?
Solved Issue
How to encrypt the password?
Unsolved Issue
4
Use a library to check the password strength!
Decision
Do not check the security of the password!
Alternative
The users are assisted in choosing strong passwords that cannot be cracked.
Pro-Argument
Implement password strength checking
FGRE-3 (Task 2)
3
The users might choose a very simple password that can be easily cracked by brute-force attacks, such as their dog's name.
Con-Argument

Figure 2: A requirement ①, development tasks ②, ③, and decision knowledge visualized as a tree. The unsolved issue ④ is highlighted in red font.

[1]https://github.com/cures-hub
[2]https://marketplace.atlassian.com/apps/1219690/decision-documentation-and-exploration

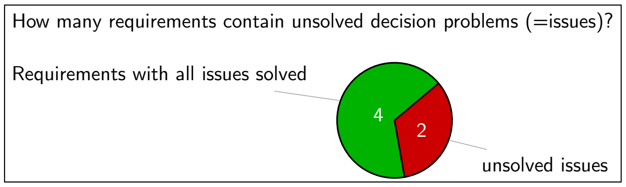
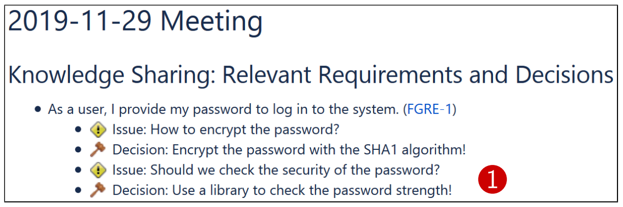


Figure 3: Pie chart to detect requirements with one or more unsolved issues as part of a dashboard.



2019-11-29 Meeting

Knowledge Sharing: Relevant Requirements and Decisions

- As a user, I provide my password to log in to the system. (FGRE-1)
  - Issue: How to encrypt the password?
  - Decision: Encrypt the password with the SHA1 algorithm!
  - Issue: Should we check the security of the password?
  - Decision: Use a library to check the password strength!

Figure 4: Automatically filled meeting agenda: The formerly open issue is now solved with a decision documented in a commit message ①.

The developers spot requirements with unsolved decision problems from a pie chart as part of a dashboard (Figure 3). This dashboard allows the developers to navigate to the requirements. The developers navigate to the respective user story and discuss the solution of the unsolved issue (Figure 2-④). They orally decide to encrypt passwords with the SHA1 algorithm but do **not** document it.

The developers implement the tasks on branches. For the first task (Figure 2-②), they create the branch *FGRE-2.pw.storage*. They make commits on this branch with the decision *Encrypt the password with the SHA1 algorithm* being part of a commit message. 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 (re)solved and the red highlighting is removed. The developers are 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.

Using the ConDec views in Jira, the requirements engineer can easily access the documented decision. To further share and discuss their knowledge, the requirements engineer creates a meeting agenda that is automatically filled with requirements and decision knowledge relevant for the current sprint (Figure 4).

3 ConDec Features

ConDec supports requirements engineers and developers in sharing their decision knowledge through an *integrated knowledge visualization* (Figure 2). Requirements engineers are supported in creating *meeting agendas* filled with relevant decision knowledge and in documenting decision knowledge in meetings. The ConDec tool support also covers 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 important decision problems and decisions regarding requirements in an *information channel*. The communication is also supported through *automated release notes* including relevant decision knowledge for the particular release.

Requirements engineers and developers exploit the integrated knowledge visualization during changes, to *estimate change impacts*. Developers can access related decision knowledge and requirements in code. Both can use various *filters*, e. g., requirements engineers could view decisions for a requirement only, without seeing the tasks and alternatives.

In order to exploit the documentation, it must be of high quality. ConDec supports *quality checking* and *enforcement* of the complete documentation of decision knowledge. On the one hand, developers can check the quality using the dashboard (Figure 3). On the other hand, the completeness of the documentation can be a quality gate in pull requests so that they can only be accepted and the respective branch can only be merged if the quality check is passed.

4 Conclusion and Future Work

ConDec supports requirements engineers and developers in sharing and exploiting decision knowledge in a lightweight, non-intrusive way. We evaluate ConDec in student projects and develop techniques to teach decision knowledge management [3]. As part of our future work, we develop methods to support decision making and decision evaluation with user feedback.

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 all the students who contribute to the ConDec tools.

References

[1] A. H. Dutoit, R. McCall, I. Mistrík, and B. Paech, *Rationale management in software engineering*. Springer, 2006.

[2] 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.

[3] 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)*, Bremerhaven, Germany, 2019, pp. 125–132.