# Experiences with Supporting the Distributed Responsibility for Requirements through Decision Documentation

Tom-Michael Hesse, Christian Kücherer, and Barbara Paech

Institute of Computer Science, University of Heidelberg, Germany
{hesse,kuecherer,paech}@informatik.uni-heidelberg.de

## 1 Introduction

In agile development projects typically all developers are responsible for requirements engineering [2]. They both elicit and shape requirements continuously. Developers *elicit* requirements from the customer. But they also *shape* requirements in discussions with the customer and within the development team. Thus, decisions are made on how to realize the requirements in the systems' architecture and implementation. This can lead to new or refined requirements.

This decision-making process requires a common language for and understanding of the elicited and shaped requirements. Also, developers need to comprehend and exploit former and current decisions. In order to address this need, several approaches propose structures and management processes for knowledge on decisions and their related requirements. However, experience reports describing such a management of decision knowledge in agile projects are rare. Therefore, we present a practical example for the management of decision knowledge. Moreover, we reflect the example to analyze our approach on decision documentation as described in [1]. In particular, we show that decision documentation was already applied in an agile project, but can be improved in order to support the effective eliciting and shaping of requirements.

## 2 Documenting Decision Knowledge

In this section, a short overview of decision knowledge and our documentation approach is given. *Decision knowledge* comprises all knowledge related to a decision problem and its solution alternatives as well as any related context information and rationales for justifying the decision. In particular, this includes links to all affected artifacts of the system, such as requirements, architecture models or code. Our documentation model structures this decision knowledge by introducing decision statements and a set of decision components as depicted in Figure 1. A *DecisionStatement* expresses the decision itself. It can be enriched iteratively by adding *DecisionComponents*, such as an *Issue* to describe a decision problem or *Alternatives* for documenting possible solutions to that issue. Moreover, context knowledge like *Assumptions* can be made explicit and rationales can be added to any element by *Arguments*.

A major advantage of our model is that there is no static template for decision description. Instead, all *DecisionComponents* can be aggregated without restrictions. As a consequence, only those components are documented which are relevant for the actual decision progress. This helps reducing the documentation effort for decision knowledge. Therefore, the model offers a valuable benefit for agile projects with less readiness to document decisions.

Moreover, the model uses general knowledge elements, which are not limited to a particular development activity like requirements engineering or architectural design. So, team members with different roles can collaborate by using the same documentation. This strengthens their mutual comprehension.

## 3 Practical Example

In this section, we investigate the demand for decision documentation and the advantages of our decision documentation model in practice. Therefore, the decision documentation of a 3 years agile project based on Scrum in the domain of electronic publishing is evaluated. In the project an interactive web-front-end for a content management system (CMS) was built with many specific requirements. Some of the main fea-

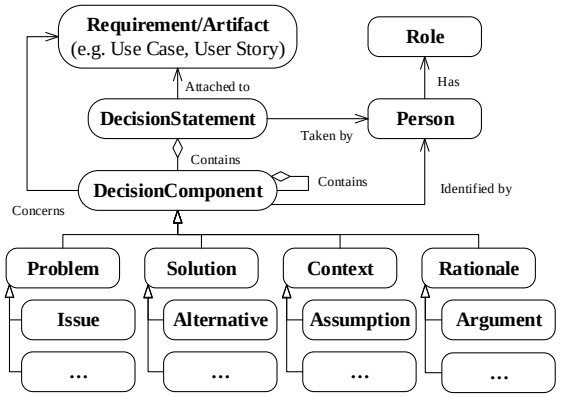


Figure 1: Knowledge Elements of the Decision Documentation Model

| Role | Tasks |
|---|---|
| Product Owner (PO) | Represent the customer, state and prioritize requirements, answer requirements-related questions from team |
| Developer (D) | Elicit requirements from product owner, elicit and shape requirements in discussions, decide on implementation of requirements |
| Architect (A) | Shape requirements in discussions, decide on the overall system structure and compliance to company policies |

Table 1: Roles and Tasks in the Example Project

tures were a powerful search among a large collection of structured documents. The team consisted of one *product owner*, one *architect* and multiple *developers* with a joint responsibility for requirements engineering. Details on their tasks are shown in Table 1.

**Need for Decision Documentation** Design issues to be decided throughout the development process were driven by requirements of multiple sources: They were stated by the product owner or resulted from the current architecture of the system and the company policies. This complexity hindered the communication and distribution of the resulting decisions. Moreover, they were hard to justify over time due to continuous change. So, the project team needed to document the decisions made. It established a lightweight documentation, the *architectural logbook*.

**Logbook Structure and Example** The logbook contained 40 entries with different sections: A header with the decisions *name* and *date*, the *involved persons*, and links to the related *requirements*. Also, the *current situation* which required a decision was described and the *decision* itself was explained. Figure 2 shows an example entry of the logbook.

**Removal of Duplicates in Hit-list**
**Date:** 22. Nov. 2013
**Involved Persons:** Alice (PO), John (D), Zoe (A)
**Requirements:** #a-182 (hit-list), #a-004 (performance).
**Current Situtation:** Due to multiple data sources, there can be the same referee displayed in the client hit list (S1). This will not match requirements of Alice (S2). Performance issues make the unification in the client difficult (S3). Scalability issues will not allow unification on the server due to chunk loading (S4).
**Decision:** Realize the merge of items within the client. Take care of performance. Implement the methods `add`, `remove` and `contains` in the hit-list classes of the datastructure. Alice is in charge of potential performance degradation but won't accept duplicate items (S5). Multiple data sources will be necessary in future, what is relevant for design (S6).

Figure 2: Anonymized and Shortened Logbook Entry

**Knowledge Structures in Use and in Demand** An analysis of the logbook showed that the aforementioned knowledge elements already can be found in the logbook entries implicitly. The given example entry contains an issue (in sentence S1), a constraint (S2), and arguments against possible solution alternatives (S3/S4). Moreover, the decision statement is explicitly given and implications of the chosen solution are documented (S5). By describing the decision a new requirement was elicited (S6). This requirement has to be considered in future decisions and their consecutive implementations.

However, a more explicit, fine-grained structure of these knowledge elements was demanded by the developers. In 19 out of 40 entries, decision and solution were described in the same sentence without considering alternatives. Moreover, 6 entries did not even contain an issue. So, developers challenged decisions made due to ambiguous or missing decision knowledge. However, the main reasons for taking the actual decision should be named explicitly to avoid that decisions made are challenged unnecessarily. In the given example, arguments against other alternatives (S3/S4) soften and thereby shape the performance requirement a-004 (S5). This relation becomes obvious when fine-grained knowledge elements are used.

## 4 Conclusion

In this paper, we have described that within agile projects product owner, developers and architects together elicit and shape requirements. Implementing these requirements leads to decisions and in consequence to new requirements or the refinement of existing ones. Documenting such decisions makes this process of eliciting and shaping requirements explicit and visible. This supports precise communication within the team and helps to justify the development progress.

Beyond the mentioned benefits of decision documentation, the continuous use and enforcement of extended structural elements allows fine-grained traceability between decision knowledge and requirements. This enhances a comprehensive and reliable use of decision knowledge in projects. So, we propose to use a structured decision documentation model. Further research should investigate a low effort integration of decision documentation structures into agile projects through an appropriate tool support.

## References


[1] T.-M. Hesse and B. Paech. Supporting the Collaborative Development of Requirements and Architecture Documentation. In *Proceedings of the 3rd International Workshop on the Twin Peaks of Requirements and Architecture (TwinPeaks'13)*, pages 22–26. IEEE, 2013.

[2] A. Sillitti and S. Giancorla. Requirements Engineering for Agile Methods. In A. Aurum and C. Wohlin, editors, *Engineering and Managing Software Requirements*, pages 309–326. Springer, 2005.