How to Improve Decision Documentation in Software Evolution?

Tom-Michael Hesse, Barbara Paech
University of Heidelberg
Heidelberg, Germany
{hesse, paech}@informatik.uni-heidelberg.de

Tobias Roehm, Bernd Bruegge
Technische Universität München
Munich, Germany
{roehm, bruegge}@in.tum.de

Abstract: This problem statement describes the lack of effective methodologies and tool support for documenting decision knowledge during software evolution. After a brief description and definition of decisions in software evolution, we outline the current mismatch between the need for decision documentation and the effort required for documentation.

1 Introduction and Definitions

Many decisions are made during the entire lifecycle of complex software systems. Those decisions can concern the project or the system and give direction to all areas of development. For instance, it is decided which requirements to realize, which architecture to implement or which milestones and work items to plan. *Decisions* at least comprise a set of alternatives and criteria to evaluate each alternative [NR05]. But they often become much more complex, as alternatives and criteria are related to particular project and system context aspects. Examples are constraints from requirements or assumptions made when analyzing the decision problem. Moreover, similar decision problems can be decided in different ways leading to different outcomes. They depend on the experience and personality of the involved stakeholders and the available time and resources. For instance, a decision can be made *naturalistic*. This means matching the given situation to former ones and apply a solution that already succeeded before. Another way is deciding by *rational* choice and evaluate all available alternatives in detail. Due to these differences, decisions are usually intertwined with *rationales* justifying them.

The underlying project and system context of decisions is likely to evolve over time. Within this evolution process, former decisions are challenged. They need to be reviewed, adapted or even withdrawn, because the system shall be kept aligned with the changing context. In order to reconsider those previous decisions, developers must retrieve and understand the related knowledge such as assumptions, alternatives or outcomes. Therefore, decisions should be captured and documented in a structured way.

2 Problem Description

However, capture and documentation of decision knowledge often is not performed at a satisfying level. A major reason was identified by a survey by Tang et al., who asked 127 practitioners for their experience with design decision documentation [TBGH06]. They found that designers agree in the benefits from documented decisions and their rationales, but miss methodologies and tool support for documentation. This is particularly true for context knowledge like possible decision downsides. Whereas 61.7% of all study participants agree in the importance of this knowledge, only 35.8% document it.

While documentation of decision knowledge is appreciated by project staff during evolution, it is not performed sufficiently during development. So, decision knowledge erodes over time and can even vaporize completely, if it remains implicit [JB05]. In consequence, understanding and reflecting previous decisions is hindered, what is a fundamental problem for the quality of future decisions. This fundamental problem can be subdivided into three questions regarding capture, structure and usage of decision knowledge (given on first list level) and their detailed contents (given on second list level):

1. How can decision knowledge be captured with as minimal effort for developers as possible?
    - How can documentation support for naturalistic decisions in software development be realized? How can this knowledge be transformed into rational decision making models? For instance, which parts of decision knowledge require manual or automatic capture?
2. Which entities, relations and attributes are best suited to structure and represent decisions in project and system knowledge?
    - Which future uses require which entities or relations? For instance, does risk assessment for decisions require documented assumptions?
3. How can captured decision knowledge be exploited in order to support the systems' evolution?
    - What kind of knowledge presentation or aggregation is useful to support which activity? For instance, which knowledge aggregation of decisions can support change impact analysis, project risk management or software maintenance?

References

[JB05] A. Jansen and J. Bosch. Software Architecture as a Set of Archtectural Design Decisions. In *5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*, pages 109 – 120, 2005.

[NR05] T. Ngo and G. Ruhe. *Engineering and Managing Software Requirements*, chapter Decision Support in Requirements Engineering. Springer, 2005.

[TBGH06] A. Tang, M. A. Babar, I. Gorton, and J. Han. Survey of Architecture Design Rationale. *Journal of Systems and Software*, 79 (12):1792 – 1804, 2006.