Architecture Decision-Making in Support of Complexity Control

Andrzej Zalewski and Szymon Kijas

Warsaw University of Technology,
Institute of Automatic Control and Computational Engineering
a.zalewski@ia.pw.edu.pl, s.kijas@elka.pw.edu.pl

Abstract. The main challenge of software engineering has always been to bring software complexity under control. Different kinds of abstractions have been devised and applied for that purpose at different levels of software design. Some of them have proven successful, such as function hierarchies, layers, API's, abstract classes, encapsulation, interfaces etc. and are widely used in practice. Concepts from the genre of software architecture should also help to manage software complexity. We argue that, before architecture decisions and architecture decision-making become a common industrial practice, they have to support software complexity management much more efficiently than at present. Despite the substantial progress already made, it is still a major challenge both in theory (architecture decisions representation and architecture decision-making methods) and practice (tool support).

Keywords: architecture decisions, architecture decision-making, architectural styles, software complexity.

1 Introduction

Complexity has always been a primary concern of software engineering. Its ultimate objective was to overcome software complexity with software design methods, models, approaches and programming paradigms. Brooks, in his famous paper [1], has called them "silver bullets" and argued pessimistically that there are no such bullets in sight. The discussion over bullets killing software complexity seems to have faded out now. We have made tremendous progress in software development methods and tools over the last fifteen years: software we could not even imagine in 1996 can be created nowadays in just a couple of hours.

This does not mean that "silver bullets" have been found. We have rather managed to transform a complexity werewolf into a genie, and to devise means of keeping him in a tightly sealed bottle of abstractions such as API's, interfaces, layers or styles/patterns [9]. Hence, we have devised numerous successful abstractions that allow us to successfully manage software complexity. This is why these most popular abstractions are widely applied in practice.

The concept of architecture decisions (AD) [2], [3] and architecture decision-making have not achieved a maturity level similar to the abstractions mentioned above, and are still far from being everyday industrial practice. The main

M. Ali Babar and I. Gorton (Eds.): ECSA 2010, LNCS 6285, pp. 501–504, 2010.

challenge to be met in order to achieve industrial maturity of architectural decision-making, is to transform it into an efficient means of managing software complexity. In the conclusion of the paper we try to envisage future developments in the area of AD making aimed at meeting this general challenge.

2 Does AD Making Help to Control Software Complexity?

Architecture decisions (AD) have been conceived as a model of software architecture alternative to the views [4]. Software modelling has always been about abstracting from details that are unimportant at a given level of abstraction. Software architecture plays the same role.

ADs successfully capture knowledge that usually evaporates during system design and evolution. However, the question of whether architectural decision-making helps to overcome software complexity has not even been raised yet. We think that in current state-of-the-art, architectural decision-making contributes very little to overcoming design complexity, while it introduces an additional complexity of its own. These limitations of architectural decision-making arise mainly from: the textual form of ADs documentation; the diversity of abstraction levels of ADs, accompanied by insufficient and often ambiguous classifications; extremely complex structures of relations between ADs and a lack of guidelines on how to properly shape these relations in the decision-making process.

2.1 Representation of Architectural Decisions

Architecture decisions are still represented as text records [3], [5], [6], certain prototype tools link ADs with some illustrating diagrams, see [7] for example. The weaknesses of textual documentation have long been identified and are a kind of mantra of software engineering: incompleteness, inconsistency, ambiguity, inefficiency in representing and sharing engineering concepts. ADs have inherited these weaknesses in full.

Diagrammatic representations of ADs could help resolve this problem, as they did in the case of structured and object-oriented analysis and design methods. However, it is very difficult to express the heart of ADs graphically, as only existence decisions [6] can easily be linked to a specific element of software design, while most ADs describe certain properties, design assumptions or constraints.

2.2 Abstraction Level and Classifications of Architectural Decisions

ADs comprising certain software architecture usually concern different levels of abstraction, different levels of architectural scope or detail. Most influential classifications by Kruchten [6] (existence, non-existence, property and management ADs) and Zimmermann [8] (executive, conceptual, technology, vendor asset ADs) substantially help to navigate through a set of ADs. However, these categories are not always precise, and in many cases can confuse engineers (consider, for example, the decision of implementing access to data via web services - engineers

can treat it either as a management (technology decision) or as an existence decision). The same category can contain decisions concerning different levels of abstraction, impacting different sets of design elements or engineering artefacts. The means of creating hierarchies of ADs (to hide unimportant details) or of aggregating ADs defining the same design element have not been devised yet.

2.3 Relations between Architecture Decisions and the Architecture Decision-Making Process

Relations between architecture decisions are used both to represent the architectural decision-making process and to supplement software architecture modelled as a set of ADs. Kruchten in [6] indicates ten kinds of relations between ADs, a number exceeding the famous 7 ± 2 rule. Thus, it could take some time to learn how to recognise each of these kinds.

ADs often represent a cross-cutting concern and can potentially be related to many other ADs (e.g. certain ADs can constrain many other ADs). This usually leads to the extremely complex structures of such relations - compare, for example, figures in [6] - pages 51-54. At the same time, there are scarcely any clues on how to shape the relations between ADs, which relations and under what conditions should or should not be modelled.

Sets of ADs do not comprise a uniform set of states similar to those known from the decision-making theory. This makes tree or graph representation of the architectural decision-making process difficult to achieve, as trees and graphs are most suitable for representing transitions between states of the same structure. This explains why design decision-making models, although developed for at least the past fifteen years [8], [10], [11], have not yet become popular in software industry. At the same time, existing tree or graph representations of architectural decision-making lack the ability to represent ADs hierarchically.

3 Challenges for Architecture Decision-Making

Architectural decision-making does not sufficiently address the concerns of software complexity control. This makes its "value proposition" disputable and will limit the transfer of the approach to the industry. We do think that the success of architectural decision-making depends mainly on its transformation into an efficient means of coping with software complexity, as in case of API's, layers, interfaces, architectural styles or design patterns.

To meet the above challenge, the following advances are needed:

- modelling ADs: developing models of ADs representing them in terms of engineering artefacts easily comprehensible for software engineers,
- organisation of ADs:
  - extending techniques of classification and clustering of ADs to improve complexity management. This should aim at the identification of ADs at a given level of abstraction, concerning certain design elements as well as clustering ADs with the structures recognised as contributing to complexity control, e.g. hierarchies or layers;

  - developing means of aggregating ADs comprising a certain design element;
  - developing means of defining and managing links between ADs and software engineering artefacts.
- structuring architectural decision-making process and providing more efficient ways of managing relations between ADs:
  - minimising the number of relations between ADs captured during the architectural decision-making process to a reasonable minimum defined by architectural decision-making methods,
  - extending existing architectural decision-making approaches to minimise dependencies between ADs by defining more precise paths of architectural decision-making while preserving the necessary level of design freedom,
  - developing predefined structures of relations between ADs, probably for chosen application domains.

References

1. Brooks, F.P.: The Mythical Man-Month: Essays on Software Engineering, 2nd Anniversary edn. Addison-Wesley Professional, Reading (1995)
2. Bosch, J., Jansen, A.: Software Architecture as a Set of Architectural Design Decisions. In: WICSA 2005, pp. 109–120. IEEE Computer Society, Los Alamitos (2005)
3. Tyree, J., Akerman, A.: Architecture Decisions: Demystifying Architecture. IEEE Software (2005)
4. Kruchten, P.: The 4+1 View Model of Architecture. IEEE Software 12, 45–50 (1995)
5. Harrison, N.B., Avgeriou, P., Zdun, U.: Using Patterns to Capture Architectural Decisions. IEEE Software 24(4), 38–45 (2007)
6. Ali Babar, M., et al.: Architecture knowledge management. Theory and Practice. Springer, Heidelberg (2009)
7. Capilla, R., et al.: A Web-Based Tool for Managing Architectural Design Decisions. In: Proc. SHARK 2006, Software Eng. Notes. ACM SIGSOFT, vol. 31(5) (2006)
8. Zimmermann, O., et al.: Managing architectural decision models with dependency relations, integrity constraints, and production rules. Journal of Systems and Software 82(8), 1249–1267 (2009)
9. Avgeriou, P., Zdun, U.: Architectural patterns revisited - a pattern language. In: 10th European Conference on Pattern Languages of Programs (EuroPlop 2005), Irsee, Germany (2005)
10. Ran, A., Kuusela, J.: Design decision trees. In: Eighth International Workshop on Software Specification and Design, pp. 172–175 (1996)
11. Workshop summary: Patterns for decision-making in architectural design, Conference on Object Oriented Programming Systems Languages and Applications, pp. 132-137 (1995)