Building Up and Reasoning About Architectural Knowledge

Philippe Kruchten[1], Patricia Lago[2], and Hans van Vliet[2]

[1] University of British Columbia
Vancouver, Canada
pbk@ece.ubc.ca
[2] Vrije Universiteit
Amsterdam, the Netherlands
{patricia, hans}@few.vu.nl

Abstract. Architectural knowledge consists of architecture design as well as the design decisions, assumptions, context, and other factors that together determine why a particular solution is the way it is. Except for the architecture design part, most of the architectural knowledge usually remains hidden, tacit in the heads of the architects. We conjecture that an explicit representation of architectural knowledge is helpful for building and evolving quality systems. If we had a repository of architectural knowledge for a system, what would it ideally contain, how would we build it, and exploit it in practice? In this paper we describe a use-case model for an architectural knowledge base, together with its underlying ontology. We present a small case study in which we model available architectural knowledge in a commercial tool, the Aduna Cluster Map Viewer, which is aimed at ontology-based visualization. Putting together ontologies, use cases and tool support, we are able to reason about which types of architecting tasks can be supported, and how this can be done.

1 Introduction

Software that is being used, evolves. For that reason, quality issues like comprehensibility, integrity, and flexibility are important concerns. For that reason also, we not only bother about today's requirements during development but also, and maybe even more so, about the requirements of tomorrow.

This is one of the main reasons for the importance of software architecture, as for instance stated in Bass *et al.* [1]: a software architecture manifests the early design decisions. These early decisions determine the system's development, deployment, and evolution. It is the earliest point at which these decisions can be assessed.

There are many definitions of software architecture. Many talk about components and connectors, or the 'high-level conception of a system'. This high-level conception then is supposed to capture the 'major design decisions'. Whether a design decision is major or not really can only be ascertained with hindsight, when we try to change the system. Only then it will show which decisions were really important. A priori, it is

C. Hofmeister et al. (Eds.): QoSA 2006, LNCS 4214, pp. 43 – 58, 2006.

often not at all clear if and why one design decision is more important than another one [9].

Architectural design, even well documented according to all the good recipes [5, 12, 14], is only one small part of the *Architectural Knowledge* that is required to design a system, or that is needed to guide a possibly multisite development team, or that can be exploited out of a system to build the next one, or that is required to successfully evolve a system. Van Vliet and Lago have pointed rightfully that all the assumptions that were made during the architectural design, all the linkage to the environment are a key component of architectural knowledge [16, 28]. Similarly, Bosch and others have pointed out that design decisions, the tight set of interdependencies between them, and their mapping to both the requirements, needs, constraints upstream, or the design and implementation downstream are also a key component of architectural knowledge [2, 15, 25].

We can usually get at the architectural *Design* part, ultimately by reverse engineering if there was no explicit documentation. This amounts to the *result* of the design decisions, the solutions chosen, not the reasoning behind them. The *Context* and some of the *Rationale* may be partially retrieved from management documents, vision documents, requirements specs, etc. Design *Decisions* and much of the *Rationale* are usually lost forever, or reside only in the head of the few people associated with them, if they are still around.

So the reasoning behind a design decision, and other forces that drive those decisions (such as: company policies, standards that have to be used, earlier experiences of the architect, etc.), are not explicitly captured. This is tacit knowledge, essential for the solution chosen, but not documented. At a later stage, it then becomes difficult to trace the reasons of certain design decisions. In particular, during the evolution one may stumble upon these design decisions, try to undo them or work around them, and get into trouble when this turns out to be very costly if not impossible. The future evolutionary capabilities of a system can be better assessed if this type of knowledge would be explicit. We use the term *assumptions* as a general denominator for the forces that drive architectural design decisions. Just like it is difficult to distinguish between the *what* and the *how* in software development, so that one person's requirements is another person's design, it is also difficult to distinguish between assumptions and decisions. Here too, from one perspective or stakeholder, we may denote something as an assumption, while that same thing may be seen as a design decision from another perspective. As a result, we are left with:

Architectural Knowledge = Design Decisions + Design (1)

In this paper, we focus on the Design Decisions and their rationale. We distinguish four types of design decisions:

- **Implicit and undocumented:** the architect is unaware of the decision, or it concerns "of course" knowledge. Examples include earlier experience, implicit company policies to use certain approaches, standards, and the like.
- **Explicit but undocumented:** the architect takes a decision for a very specific reason (e.g., the decision to use a certain user-interface policy because of time constraints). The reasoning is not documented, and thus is likely to vaporize over time.