Structuring Software Architecture Project Memories

Remco C. de Boer, Rik Farenhorst, Viktor Clerc, Jan S. van der Ven,
Robert Deckers, Patricia Lago, and Hans van Vliet

Vrije Universiteit, Amsterdam, the Netherlands
{remco, rik, viktor, patricia, hans}@few.vu.nl
University of Groningen, Groningen, the Netherlands
j.s.van.der.ven@rug.nl
Philips Research, Eindhoven, the Netherlands
robert.deckers@philips.com
Getronics PinkRoccade, Amsterdam, the Netherlands
rik.farenhorst@getronics.com
CIBIT Consultants | Educators, Bilthoven, the Netherlands
vclerc@cibit.nl

Abstract. In order to prevent knowledge dispersion in the software architecting process, it is crucial to deploy sound mechanisms to manage and share vital architectural knowledge. Continuous organizational learning is an important vehicle to address this issue. We present a model that provides a generic structure for software architecture project memories. Key concepts in this model are design decisions and their rationale. This focus ensures that not only the software artifacts themselves are subject to an organization's knowledge management efforts, but also the technical, organizational, political and economic influences on these artifacts. In our current work, we empirically validate the model presented in this paper. Case studies are being performed in which we forward engineer (capture) and reverse engineer (recover) architectural knowledge by means of the presented model.

Introduction

This paper presents a model for structuring software architecture project memories. This architectural knowledge model captures the interdependencies between architectural design decisions [1,2], architectural descriptions [3] and the various stakeholders of the system, and treats these design decisions as first-class entities. A network of design decisions can be instantiated based on this meta-model. Combined with the associated design artifacts, this network could then serve as a project memory containing vital architectural knowledge, including the now oft-lacking architectural know-why.

A software architecture embodies the early design decisions [4]. These early decisions shape the systems' development, deployment, and evolution. Software architecture usually addresses solution aspects like software artifacts and the technology being deployed. We tend to leave technical, business, organizational, political, and economic drivers that influence these artifacts implicit. Nevertheless, these drivers are

an important part of the architectural knowledge in a software development project. Leaving such drivers implicit results in poor accessibility of the organization's architectural body of knowledge. A possible solution to this problem is making the architectural knowledge explicit by capturing it. Without capturing existing knowledge, additional (and not exactly quantifiable) costs are spent each time knowledge has to be transferred within the organization. People have to rethink the reasons why certain decisions have been made, or why errors occurred, and they have to repeatedly follow the same paths to achieve similar results. Furthermore, explicit knowledge can be shared more easily between different stakeholders. Therefore, if applicable, it can be reused in different contexts, such as different development projects, application domains, or development sites.

The benefit of a software architecture project memory based on explicit, or codified, knowledge depends in particular on the level of knowledge reuse in the organization [5]. Since architectural knowledge comprises the earliest knowledge in a software development project, it is likely to be reused often.

Although in general one should strive to capture as much relevant knowledge as possible, not all organizational knowledge can be made explicit. Tacit knowledge within an organization can only be made accessible through pointers to the members that hold that knowledge [6]. Besides re-using captured knowledge, learning from past experiences on an organizational level is of utmost importance in modern software development. This is in particular the case for software architecture, since the architecture embodies the early design decisions. Such learning needs can be addressed by systematic application of organizational learning principles [7], supported by organizational (project) memories [8].

This paper is organized as follows. First, we re-iterate the value of knowledge management in Software Engineering (SE), and identify the importance of a first-class representation of design decisions. Next, our model for architectural knowledge is presented. To show the practical value of the model, we discuss some examples of project memory instantiations based on the presented model. In the final section, we present our conclusions and discuss our current and future work on application of the model in industrial settings.

Related Work on Knowledge Management in SE

Over the past decade, the notion of Learning Software Organizations (LSO), i.e. organizations that improve and extend their software engineering knowledge through continuous learning and exchange of experience (e.g. by means of an Experience Factory [9]), has gained considerable interest from both the research and industrial communities.

In order to be able to learn from experience, a project memory is necessary. Such a memory can be supported by various SE activities, like version control, change management, documentation of architectures and design decisions [10], and requirements traceability [11]. Learning from and improvement of the software development process can be supported by predictive models that guide decision making based on past projects [12]. The idea of guiding decisions is also present in [13], in which Software

Engineering Decision Support is presented as an extension to the LSO approach. Decision guidance also needs the knowledge incorporated in a project memory.

In general, different types of knowledge should be present in a project memory: know-how, know-why and know-what [6]. Existing notational and documentation approaches to software architecture typically focus on the components and connectors. These approaches fail to document the design decisions embodied in the architecture as well as the rationale underlying the design decisions. This means that often the architectural knowledge present in an SE project memory adequately covers know-what and know-how, but fails to address the know-why of a software architecture.

Failure of addressing know-why results in high maintenance costs, high degrees of design erosion, and lack of information and documentation of relevant architectural knowledge. The know-why - or rationale - of an architecture manifests itself through the design decisions that are incorporated in the architecture. It is for this reason that recent research efforts in software architecture focus on architectural design decisions and their rationale [1,14,15]. Since the design decisions lead to the design, software architecture can be viewed as the collection of these decisions [16], or as the design decisions plus the resulting design [2]. Existing methodologies for capturing rationale, such as IBIS [17] and its descendents, tend to view the decision process separately from its environment, e.g. the software architecture. In order to effectively manage know-why as part of architectural knowledge, viewing software architecture through architectural descriptions only (e.g. [3]) is insufficient. We need to go beyond this view and include architectural design decisions as first-class entities.

A Model for Architectural Knowledge

A conceptual schematic outline of the architecting process is depicted in Fig. 1. This figure shows the stakeholders as being central to the architecting process. This corresponds to the notion that a software architecture should be stakeholder-driven. The stakeholders, in particular the architects, take decisions according to their roles. These decisions can be influenced by environmental events, i.e. events that are relevant but in principle do not belong to the software architecting and development realm, such as budget constraints. The decisions are reflected in design artifacts and architectural descriptions, e.g. documented views, but also 'lower-level' artifacts such as source code. A stakeholder responsible for an artifact changed due to a decision might find a need to take subsequent decisions. The responsibilities are defined by the stakeholders' role(s). The schematic outline in Fig. 1 shows the interdependencies between architectural design decisions and design artifacts. In managing the architectural body of knowledge, the five elements from Fig. 1 must be taken into account. Such a body of knowledge contains both tacit and explicit knowledge. This means that the project memory must not only provide explicit (captured) architectural knowledge, but also pointers to stakeholders that possess tacit knowledge on certain subjects.

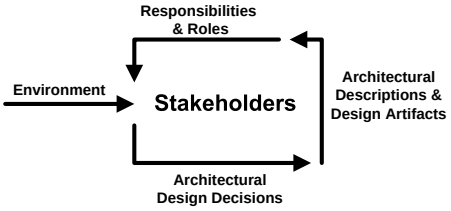

Fig. 1. Schematic outline of the architecting process. The concepts from this Figure are addressed in detail in the model from Fig. 2.

At a meta-level we want to structure software architecture project memories in such a way that it is clear what can exist and what can happen during the architecting phase of a software development project. This structure must remain at a high level of abstraction, since the instantiations of elements in a project memory can vary case by case. Based on the outline in Fig. 1, the model in Fig. 2 provides an abstract view of the architectural knowledge domain, which allows for clean reasoning about the needs for managing architectural knowledge. To this end, it provides a clear set of concepts, which facilitates communication and sharing of architectural knowledge.

The model in Fig. 2 again shows 'Stakeholder' as a central concept to the software architectural knowledge domain. The concepts of *Stakeholder*, *Concern*, *Viewpoint* and *View* coincide with the definitions of these terms in IEEE-1471 [3]. *Role* and *Activity* signify the same concepts from SPEM [18], while *Artifact* coincides with the SPEM WorkProduct[1]. Our model thus builds on established standards in the field.

In our model, *Concern* is augmented with one or more *Decision topics*. These are the topics that need to be addressed in the decision making process due to a concern a stakeholder has. Decision making is viewed as proposing and ranking alternatives, and selecting the alternative that has the highest rank. The chosen alternative becomes the decision. The alternatives that are proposed must address the decision topic, and can be ranked according to how well they satisfy this and other concerns (e.g. quality attributes). The model shows *Architectural Design Decisions* as those decisions that can be reflected in the architectural design. The *Architectural Design* is the result of all architectural design decisions and other architectural drivers. Any *Force*, i.e. any decision or external event, can become an *Architectural driver* when someone assumes the force has influence on the design. When the influence is not only assumed, but even enforced[2], this results in new concerns for the stakeholder that is responsible for the architectural models that capture that part of the design that is influenced by the driver.

[1] In our view, Artifact is a better known and widely accepted concept in Software Engineering.
[2] This is a subtle difference between the *recognition* of an architectural driver and the actions *because of* this driver.

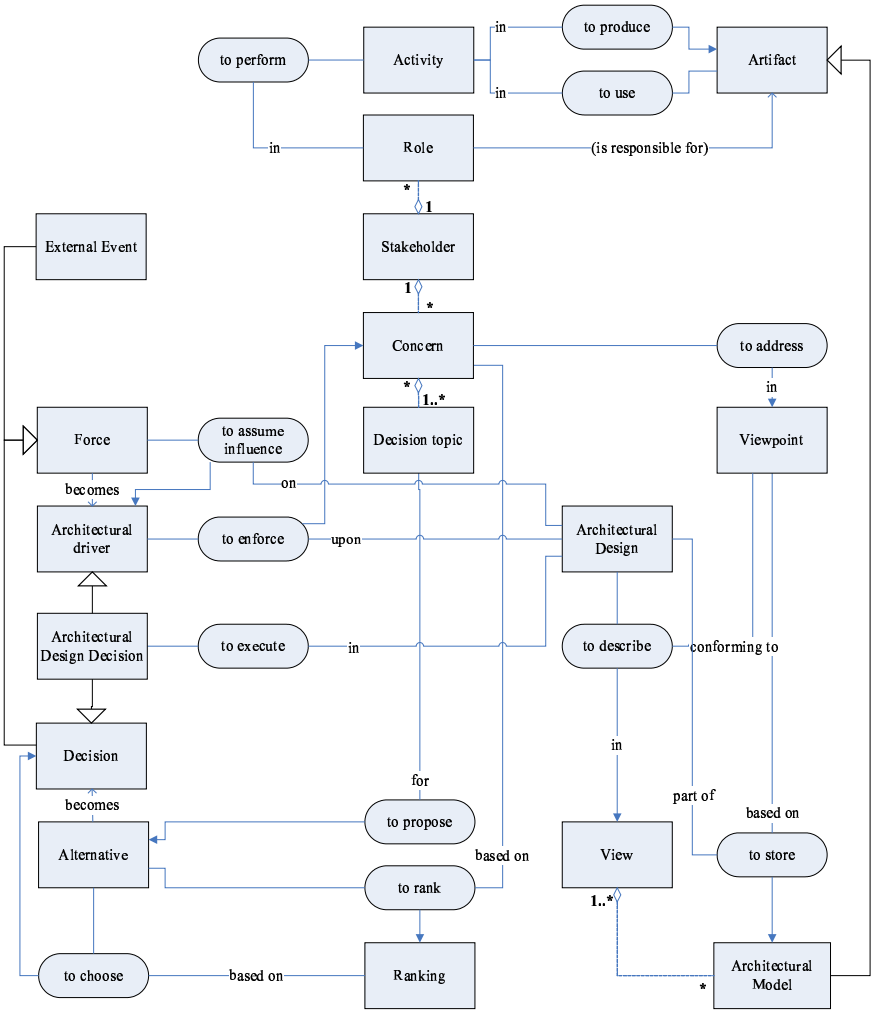

Fig. 2. Model for Architectural Knowledge. The rectangular elements represent entities; the elliptical ones denote actions on or with these entities. Arrows indicate the creation of new instances of an entity. Dashed lines represent attribute relationships. Sub-/superclass relationships are represented as generalizations [19]. Actions and entities that are linked together can be read as sentences, e.g "Someone chooses an alternative based on a ranking which results in a decision".

Since the rationale for an architecture manifests itself through the architectural design decisions, this model captures the architectural know-why. The trajectory from concern and decision topic through alternative and ranking to decision provides insight into the rationale of a decision taken, and hence into the rationale of the architecture.

Pointers to tacit architectural knowledge can be found by examining the actors of the actions present in the model. This knowledge can be captured, by storing not only the role in which an activity has been performed, but also information about the stakeholder that acted in this role. Knowing e.g. who took which decision on a specific decision topic can guide someone in need for knowledge about that topic to the right person.

Towards Software Architecture Project Memories

The model in Fig. 2 remains at a high level of abstraction. This enables organizations that base their software architecture project memory on this model to shape, or specialize, the model to their organization-specific needs. The organization can then instantiate a software architecture project memory based on this specialization. Concepts, terminology, and processes used within the organization can be mapped onto the generic concepts from the model.

An example specialization can be found in [20]. In that article, an ontology is presented that distinguishes four kinds of architectural design decisions. An organization that wants to organize their design decisions according to this ontology can extend the concept of Architectural Design Decision from our model with the different decision types from the ontology. Conceptually, this entails a specialization of the Architectural Design Decision class by four subclasses.

Specialization of the model is also used in a case study we are currently performing with a large Dutch IT-service organization. This organization uses a methodology to guide the architecting process. This methodology centers around four key concepts: targets, principles, frameworks, and models. These concepts differ in granularity and scope of the associated decision topics. Goals - descriptions of what one wants to accomplish - are made more specific in principles - statements on the desired implementation of a goal. Frameworks are even more specific, and offer methods, techniques, standards, tools, and guidelines as a specific implementation of one or more principles. Finally, models exhibit a schematic architectural design that adheres to all applicable frameworks and principles. In this methodology, all but the models are instances of architectural design decisions. The methodology's 'models' are architectural models that represent (part of) the architectural design (see Fig. 2). In an architectural project memory for this organization, we must be able to distinguish between the different types of decisions in order to conform to the organization's methodology. Since the decision types depend on the granularity and scope of the associated decision topics, we can use a classification of decision topics for this distinction. This is an extension to the model, not unlike the design decision ontology described above.

Within this organization, a tool is available that supports the architects' decision making. The tool consists of a knowledge base, containing design decisions, guide-

lines, best practices and the like. The tool intentionally covers only part of the decision making, leaving a large number of decisions open. Observations during our case study indicate that this tool is not used as often as intended, which is reflected in various knowledge (re)use issues within the organization. This has partly to do with a lack of knowledge-feedback when the remaining decisions are taken, partly with the lack of structure of the knowledge captured. Using our model, we are working on a solution by a) specializing our model to this organization's specific needs and b) using this extension to (re)structure the knowledge in their knowledge base. In addition, the extension allows us to codify the remainder of the architectural decision making process without the need for changing the architecting methodology used, enabling the essential knowledge-feedback loop currently missing.

Conclusions and Outlook

Current documentation and notational approaches to software architecture typically focus on the components and connectors, and leave the know-why - or rationale - of an architecture implicit. On the other hand, methodologies for capturing rationale tend to view the decision process separately from its environment. Failure to adequately manage know-why results in high maintenance costs, high degrees of design erosion, and lack of information and documentation of relevant architectural knowledge.

The architectural knowledge model presented in this article provides a high-level model for structuring software architecture project memories. Instead of merely focusing on either artifacts or decisions, our model provides a holistic view on architectural knowledge. It associates know-why, or rationale, with the know-what and know-how contained in design artifacts. Instantiations of this model can be tailored to reflect the specific needs of an organization.

Our current work focuses on the application of our model in various industrial settings. To this end, we are conducting a number of case studies, one of which has been briefly outlined in the previous section. In the case studies, we conduct interviews with various producers and consumers of architectural knowledge, including architects, reviewers, and developers. We explore the possible and existing uses of architectural knowledge within these organizations, and how to support this use with instantiations of our model.

Acknowledgements

This research has been partially sponsored by the Dutch Joint Academic and Commercial Quality Research & Development (Jacquard) program on Software Engineering Research via contract 638.001.406 GRIFFIN: a GRId For inFormatIoN about architectural knowledge. We would like to thank Dieter Hammer and Paris Avgeriou (University of Groningen) for their valuable input.

References

1. Tyree, J., Akerman, A.: Architecture decisions: Demystifying architecture. IEEE Software 22(2) (2005) 19{27
2. Kruchten, P., Lago, P., Vliet, H.v.: Building up and reasoning about architectural knowledge. In: 2nd International Conference on the Quality of Software Architectures (QoSA 2006) (Under Submission)
3. IEEE: IEEE recommended practice for architectural description of software-intensive systems. Standard 1471-2000, IEEE (2000)
4. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. Second edn. SEI Series in Software Engineering. Addison-Wesley Pearson Education (2003)
5. Hansen, M.T., Nohria, N., Tierney, T.: What's your strategy for managing knowledge? Harvard Business Review 77(2) (1999) 106-116
6. Ruhe, G., Bomarius, F.: Enabling techniques for learning organizations (1999)
7. Dings_yr, T., Lago, P., Vliet, H.v.: Rationale promotes learning about architectural knowledge In: 8th International Workshop on Learning Software Organizations (LSO 2006)
8. Altho_, K.D., Bomarius, F., Tautz, C.: Knowledge management for building learning software organizations. Information Systems Frontiers 2(3/4) (2000) 349-367
9. Basili, V.R., Caldiera, G., Rombach, D.H.: The experience factory. In: Encyclopedia of Software Engineering. Volume 2. John Wiley & Sons Inc. (1994) 469-476
10. Moran, T.P., Carroll, J.M., eds.: Design Rationale: Concepts, Techniques, and Use. Computers, Cognition, and Work. Lawrence Erlbaum Associates, Inc. (1996)
11. Ramesh, B., Jarke, M.: Toward reference models for requirements traceability. IEEE Trans. Softw. Eng. 27(1) (2001) 58-93
12. Rus, I., Lindvall, M.: Knowledge management in software engineering. IEEE Software 19(3) (2002) 26-38
13. Ruhe, G.: Software engineering decision support - a new paradigm for learning software organizations. In Maurer, S.H., F., eds.: LSO 2002. Volume 2640 of Lecture Notes in Computer Science., Springer-Verlag (2002) 104-113
14. Bosch, J.: Software architecture: The next step. In: Software Architecture: First European Workshop (EWSA 2004). Volume 3047 of Lecture Notes in Computer Science. Springer-Verlag GmbH (2004) 194-199
15. Ven, J.S.v.d., Jansen, A.G.J., Nijhuis, J.A.G., Bosch, J.: Design decisions: The bridge between rationale and architecture. In: Rationale Management in Software Engineering. Springer-Verlag (2006)
16. Jansen, A., Bosch, J.: Software architecture as a set of architectural design decisions. In: 5th IEEE/IFIP Working Conference on Software Architecture (WICSA 2005), Pittsburgh, Pennsylvania, USA (2005)
17. Kunz, W., Rittel, H.: Issues as elements of information systems. Technical Report Working Paper 131, University of California (1970)
18. OMG: Software process engineering metamodel specification. Technical Report formal/05-01-06, Object Management Group (2005)
19. OMG: Unified modeling language specification. Technical report, Object Management Group (2005)
20. Kruchten, P.: An ontology of architectural design decisions in software-intensive systems. In: 2nd Groningen Workshop on Software Variability Management, Groningen, NL (2004)