Towards Using Architectural Knowledge

Paris Avgeriou
University of Groningen
The Netherlands
paris@cs.rug.nl

Patricia Lago
VU University
Amsterdam
The Netherlands
patricia@cs.vu.nl

Philippe Kruchten
University of
British Columbia
Vancouver, Canada
pbk@ece.ubc.ca

DOI: 10.1145/1507195.1507219
http://doi.acm.org/10.1145/1507195.1507219

Abstract

The third workshop on Sharing and Reusing Architectural Knowledge (SHARK) was held jointly with ICSE 2008 in Leipzig, Germany. It featured two keynote talks, thirteen research position statements and three working groups that discussed on focused topics. This report presents the themes of the workshop, summarizes the results of the discussions held, and suggests some topics for future research.

Introduction

Software architecture plays an important role in managing the complex interactions and dependencies between stakeholders and serves as a reference artifact that can be used by stakeholders to share knowledge about the design of a system. Architecture also facilitates early analysis of the system, especially with respect to quality attributes and maintainability of the system. Current approaches of software architecting focus heavily on documenting components and connectors and fail to document the design decisions that *produced* the architecture – as well as the organizational, process and business rationale underlying those design decisions. This lack of relevant architectural knowledge and documentation can negatively impact maintenance costs and lead to architectural erosion and mismatch. The SHARK 2008 workshop focused on current approaches that tackle this problem: methods, languages, and tools that can be used to extract, represent, share, apply, and re-use architectural knowledge.

Architectural Knowledge (AK) is defined as the integrated representation of the software architecture of a software-intensive system or family of systems along with architectural decisions and their rationale, external influence and the development environment.

The Keynote Talks

The two keynote speakers work on the domain of Requirements Engineering and were asked to present their viewpoints on AK in a way that bridges the problem and the solution domain. Professor Axel van Lamsweerde from the University of Louvain, Belgium talked about Goal Models as a type of Architectural Knowledge. He started from the typical intertwining between Requirements

Engineering (RE) and Architecture Design (AD) [1], and discussed how they both involve making decisions by selecting among alternatives. His thesis was to use goals as a means to bridge the gap between RE and AD. He distinguished between different types of AK and categories of design decisions and demonstrated how late RE involves selecting between alternative options, thus making early AD decisions. Finally, he presented a concrete way to derive specific architecture solutions (dataflow architecture views, selection of architectural styles, recursive refinement) starting always from a number of goals (architectural, functional and non-functional). With this approach, RE and AD take place simultaneously in a systematic manner, with architectural structures explicitly linked with goals (and therefore the rationale).

The second keynote speaker, Dr. Jon Hal, from the Open University, UK, presented Problem-Oriented Engineering [2]: an approach that tackles the challenges of design by exploring both the problem and the solution space. He discussed the notion of (software) design, and how it is driven by the relation between problem and solution, as well as the argument (e.g. rationale) for both. Essentially designing involves recursively developing the problem and its argument, developing the solution and its argument, and finally relating the problem and solution again with an argument. The argument comes from validating both the problem and the solutions, as well as their match. He went on to discuss the challenges in both finding and validating problems and solutions and some established techniques for each case. He also discussed several challenges that arise during this process like solving unfamiliar problems, satisfying quality requirements, making tradeoffs, facing problems that come in groups etc.

Working Group Discussions

The workshop accepted 13 research and position papers[1] for inclusion in the proceedings. The papers can be divided into three distinct categories: software patterns as means to codify architecture knowledge in different domains (e.g., SOA, Global Software Development); extending the traditional software architecting process with general or specific reusable knowledge (e.g., in selecting connectors or in evolving the system); aspects of using AK in research and practice. The authors of accepted papers were invited to present their ideas to the workshop in the form of a position statement. The presentations[2] of the accepted papers provided the basis for further dialogue among the workshop participants in several working group sessions. The topics selected for further discussion were:

- The role of software patterns in creating, sharing and using AK
- Design Rationale and AK
- AK in support of software evolution, in particular in the context of software product lines

The following sections elaborate on the results of the discussions in the working groups.

[1] Papers accepted for the SHARK 2008 workshop are available at the ACM Digital Library

[2] PDF versions of the presentation slides are available at the SHARK wiki http://www.cs.rug.nl/shark/

Software Patterns and AK

The workshop had a significant number of papers related to the use of software patterns, which is not surprising: the patterns community has always advocated sharing of generic knowledge in the form of patterns as small, digestible chunks. The working group started by discussing an existing approach that proposes software patterns as an effective and inexpensive means to capture significant architecture decisions during the architecting process [4]. Based on this approach we decided to find a broader set of potential ways of using software patterns as generic AK, as well as the issues and challenges that lie ahead.

One of the results of the first SHARK workshop [10] was to look at AK as both a product and a process. The former concerns artifacts of AK, e.g., design decisions, domain models, languages and architecture views. The latter concerns using AK during the software development process, such as use cases, methods, tools and services. Patterns that provide knowledge on how to structure a system and focus on the functionality, behavior and quality of that system clearly belong to the first category, since the application of patterns corresponds to making decisions. On the other hand, patterns that provide knowledge on how to structure an organization and manage the software development process belong to the second category. In certain cases the distinction between the two categories is not clear; for example agile patterns concern primarily the development process but also provide knowledge about the products (e.g. code artifacts and test cases).

Table 1 shows the types of patterns that can be used as valuable AK, both as a product and as a process. The former are further categorized under the different phases of the development lifecycle. Please note that the in each category, only the most representative/seminal work is referenced; in fact there are many other sources in the literature that contain patterns for the various phases. Furthermore there are more categories of patterns in the literature than the ones listed here.

Table 1 – Patterns as Architecture Knowledge

AK as a product				
Requirements	Analysis	Architecture	Design	Implementation
Problem Frames [3]	Analysis Patterns [7]	Pattern-Oriented Software Architecture [5]	Object-Oriented design Patterns [6]	Software Factories [11]
Domain–specific Pattern Languages e.g. SOA, Enterprise Applications				
AK as a process				
Agile patterns [12]	Global software Development	Organizational patterns [13]	Open Source Software Development	

The use of patterns in both categories and their sub-categories indicates a wealthy amount of AK that can be used throughout the development lifecycle. Unfortunately the different pattern languag-

es are currently not linked to each other which inhibits a more holistic approach of using the entire corpus of patterns knowledge coherently. It also hinders traceability between the different artifacts, e.g. between requirements, architecture decisions and the implementation. One of the research challenges is to attempt to integrate the different pattern languages that span through the different lifecycle phases, in a similar fashion to Model-Driven Engineering: going from abstract to specific through refinement.

We further discussed how the patterns for software engineering processes support *using* AK. During the second SHARK workshop we had made the distinction between *codification* techniques that make AK explicit in a model or a document, and *personalization* techniques that tailor the knowledge system to specific people and organizations, keeping most AK tacit [14]. In this respect, agile process focus heavily on personalization, sharing the knowledge in a face-to-face communication. There is also typically an informal "yellow pages" directory, so that people are aware of which team member has knowledge about which aspect of the system development. Personalization techniques work very well in agile methods, as long as the total number of team members is small to medium; otherwise there may be scalability issues. On the other hand, more heavyweight process, focus mostly on codification, i.e. writing down AK into formal or semi-formal documentation and models. Some personalization is also used in heavyweight processes, as it is usually not possible to document everything or to keep the documentation up-to-date.

Finally we addressed the issue of architecture recovery, and especially how to re-create the AK that has evaporated through the use of patterns. A common practice to achieve this is through architecture retrospectives or even better architecture reviews. In both cases AK can be recovered by finding the patterns used, the specific variants of the patterns selected and potential exceptions to those patterns enforced by the system requirements. These are all conscious or subconscious decisions. The rationale behind these decisions can be found in the patterns documentation, as it usually contains a number of forces, which are balanced by the patterns and also the consequences that the patterns entail on a system. Particular attention should be paid, when a specific part of the system does not follow any pattern. Another source of information to recover AK is the sequence that the patterns have been applied, by interviewing the involved architects or designers. This indicates the sequence of decisions made, especially throughout the system evolution.

We agreed that one of the most challenging research topics in the use of software patterns as AK is the provision of intelligent support by appropriate tooling, which is currently missing. Tools should help software engineers to reuse AK in an efficient way, and at the same time document the produced AK in an unobtrusive way, i.e. without getting in the way of the natural flow of design or architecting. Such tools should not be prescriptive; rather they should actively provide suggestions giving the software engineers the freedom to make informed decisions.

On Design Rationale

The discussion focused on the major research questions on rationale behind architectural design decisions, and the answers so far.

This working group counted a balanced number of industrial and academic participants, in both architecture and requirements engineering. We agreed on the following list of remaining open research questions:

1. What is the best way to keep track of Rationale? Rationale management is not a new field; much work has been carried out in this area in the past. However we still miss effective and pragmatic *process support* to record rationale 'while doing' rather than documenting it in a 'post-mortem' additional activity. A related question is: when is the right time to capture Rationale?
2. What is the minimal set of Rationale to be recorded? If recorded, rationale should be easy to search and locate. In other disciplines, we know that quantity never ensures quality; on the contrary quantity often hinders usage (as research in the field of Software Reuse taught us). What knowledge entities about rationale are the most useful in a later stage? Some examples mentioned are: less obvious, most relevant, most complicated, most controversial, exposing risk.
3. Who is the target? A problem hindering rationale documentation is that rationale is not used now (e.g. "by me, in my current project, by my current client"), but it is meant to be used in the future, eventually by others. It is hence essential to reflect on who is the future consumer of rationale knowledge. Is it meant for the readers of the written documentation? Is this useful for peers? Or for myself? Understanding the target is further essential to determine what to record (i.e. to scope question 2.).
4. What does "documenting Rationale" essentially means? Is it about pointing to the right pieces of information, and/or eventually the right people (personalization strategy)? Or do we mean to create some kind of Rationale view?
5. Why do we need Rationale at all? Potential reasons include: to reach consensus in case of conflicts; for education/knowledge transfer; to build trust; to prioritize requirements; for impact analysis; to evolve/change decisions. While some reasons are meant to support technical/engineering activities, others support business or broader organizational objectives. Rationale behind keeping track of Rationale is of course also essential for scoping.
6. How to elicit and capture Rationale? While question 1 focuses on integration in the architecting process, here we address tool support, i.e. what technical solutions fit best in a certain way of integrating Rationale tracking in the architecting process.

In conclusion, in spite of the consistent body of work and tool support already existing in the area of rationale modeling and management, we still lack satisfactory answers to the questions above. A possibility is to abandon general-purpose approaches (which seem to get too complicated or not pragmatic enough) and try to research domain- or organization-specific solution following the questions identified.

AK and evolution of a family of software product

The discussion was focused on the role of architectural knowledge (including rationale) for effectively evolving architectures of a

single system or a family of systems. We discussed the kind of needs that architectural knowledge can serve while evolving a system's architecture, especially for performing impact analysis, ensuring architectural integrity and consistency, and making all the changes in conformance with the overall vision and purpose of the architectural decisions. The group also discussed the importance of the availability of architectural knowledge for product derivation based on the core assets of a family of products. It was discussed that rationale underpinning the variability models and identified variation points is extremely important to ease the pain during the derivation process. One of the main challenges identified were the scalability of the available approaches to capture and maintain architectural knowledge with a minimum amount of resources usually available to project teams for value-added activities like knowledge management. The group also emphasized the need for standardized or semi-standardized ways of documenting the knowledge and rationale that is absolutely necessary to support the architectural level modifications and evolutions. We also debated the role of rationale in Model-Driven Development and how the quality of the models built at different levels of abstractions and generated code can be improved by improving the management of the knowledge that underpins the models.

Acknowledgments

This workshop is part of the dissemination activities of 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 extend our thanks to all those who have participated in the organization of this workshop, particularly to the program committee, which was comprised of:

- Pierre America, Philips Research, the Netherlands
- Muhammad Ali Babar, LERO, Ireland
- Martin Becker, Fraunhofer IESE, Germany
- Jan Bosch, Intuit, USA
- Rafael Capilla, Universidad Rey Juan Carlos, Spain
- Torgeir Dingsoyr, Sintef, Trondheim, Norway
- Rich Hilliard, independent consultant, USA
- Ralph Johnson, University of Illinois at Urbana-Champaign, USA
- Axel van Lamsweerde, Universite Catholique de Louvain
- Ivan Mistrik, independent consultant, Germany
- Bashar Nuseibeh, Open University, UK
- Eltjo Poort, LogicaCMG, The Netherlands
- Antony Tang, Swinburne University of Technology, Australia
- Hans van Vliet, VU University Amsterdam, Netherlands
- Uwe Zdun, Technical University of Vienna, Austria

References

[1] Axel van Lamsweerde (2008) *Systematic Requirements Engineering - From System Goals to UML Models to Software Specifications.* Wiley.

[2] Jon G Hall, Lucia Rapanotti, and Michael Jackson (2008) Problem Oriented Software Engineering: Solving the Package Router Control problem. *IEEE Transactions on Software Engineering,* 34 (2). pp. 226-241.

[3] Michael Jackson, Problem Frames (2001) *Analyzing & Structuring Software Development Problems,* Addison Wesley Professional.

[4] Neil Harrison, Paris Avgeriou and Uwe Zdun (2007) Architecture Patterns as Mechanisms for Capturing Architectural Decisions, *IEEE Software* 24 (4), July-August 2007, pp. 38-45.

[5] Frank Buschmann, K. Henney, Douglas Schmidt, *Pattern-Oriented Software Architecture: A Pattern Language for Distributed Computing* Wiley Software Patterns Series.

[6] Erich Gamma, Richard Helm and Ralph Johnson (1995) *Design Patterns: Elements of Reusable Object-Oriented Software,* Addison-Wesley.

[7] Martin Fowler (1997) *Analysis Patterns for Reusable Object Models,* Addison-Wesley,.

[8] IEEE Std 1471:2000 – *Recommended practice for architectural description of software intensive systems.* Los Alamitos, CA: IEEE, 2000.

[10] Patricia Lago and Paris Avgeriou (2006) First ACM Workshop on SHAring and Reusing architectural Knowledge (SHARK); Final workshop report. *ACM SIGSOFT Software Engineering Notes,*31 (5), Sept. 2006, pp. 32-36.

[11] Jack Greenfield and Keith Short (2004) *Software Factories,* Wiley.

[12] James O. Coplien and Neil B. Harrison (2005) *Organizational Patterns for Agile Software Development,* Prentice Hall.

[13] David M. Dikel, David Kane, and James R. Wilson (2001) *Software Architecture: Organizational Principles and Patterns,* 1st edition, Prentice-Hall.

[14] P. Avgeriou, P. Kruchten, P. Lago, P Grisham, and D. Perry (2007) Architectural knowledge and rationale: issues, trends, challenges, SHARK workshop at the 29th Int. Conf. on Software Engineering (ICSE 2007), May 20-26, 2007, Minneapolis, *ACM SIGSOFT Software Engineering Notes,* 32 (4), pp. 41-46.