Architectural Knowledge and Rationale – Issues, Trends, Challenges

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

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

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

Paul Grisham and Dewayne Perry
University of Texas at Austin, USA
{grisham, perry}@ece.utexas.edu

Abstract

The second workshop on Sharing and Reusing Architectural Knowledge (SHARK) and Architecture rationale and Design intent (ADI) was held jointly with ICSE 2007 in Minneapolis. 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 on components and connectors but 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 2007 SHARK/ADI 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.

Working Group Discussions

The two-day workshop accepted 12 research and position papers[1] for inclusion. The authors of accepted papers were invited to present their ideas to the workshop. 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:

- Documentation of architectural decisions: what is optional and what is essential?
- Codification versus personalization strategies: what works and what does not work?
- Adopting an AK-centric approach in an organization: what is the added value for different stakeholders?
- Tool support for AK.
- Measurement and empirical evaluation of AK.

In addition, workshop participants discussed how to add AK to the revision of IEEE-Std-1471-2000. The results of the discussions are summarized in the following sections.

Documentation of architectural decisions

Based on previous work by Kruchten, Lago, van Vliet [5,6], Tyree & Ackerman [8], Dueñas & Capilla [1], this working group attempted to specify what architectural decision documentation must include, and what additional information a decision documentation can include.

Core

At a minimum, all decisions should contain at least:

- Description (what is decided)
- Issue (being solved)
- Rationale (reason this was decided)
- Discarded options

Relationships

Decisions can be related to other decisions or software development artifacts to enhance traceability using the following characteristics:

- Links to other decisions, with a relation type {forbids, enables, conflicts, *etc.*}
- Upstream and downstream traces to requirements, design, implementation, and tests
- Categories (such as keywords or aspects)

Management

Some systems could make use of additional information to manage decision or sets of decisions such as:

- Name, ID, system, author, owner, *etc.*
- Version history (date, author, delta, *etc.*)
- Status (tentative, decided, challenged, rejected)
- Decision type (according to a specific taxonomy)
- Results of cost or risk analysis.

Codification versus personalization

Codification and personalization are emerging trends in the field of software architecture and architectural knowledge [2]. *Codification* formalizes the activities aimed at making AK explicit in some model or documentation and allows those models to be interpreted and reused in a standardized way. *Personalization* is the process of tailoring a knowledge system to specific people and organizations conducting the architecting process, in sharing their experience and knowledge in-the-field. The difference between the two strategies is that, while codification wants to separate the

[1] Papers accepted for the 2007 SHARK/ADI workshop are available at the ACM Digital Library

[2] PDF versions of the presentation slides are available at http://www.cs.rug.nl/~paris/SHARK-ADI2007/

AK from its owner, personalization supports knowledge transfer dependent on the knowledge provider or consumer.

Starting with the question "What are the dos and don'ts in personalization and codification of architectural knowledge?" the working group offered its ideas on a list of best-practices and "anti-practices". As the group was composed of both researchers and practitioners, the working group could contribute with experience from both academia and industry.

The group first explored the relation between codification and personalization strategies in general terms. As illustrated in Figure 1, it is important to identify the alignment between these two types of strategies, as they are not independent. For example, organization-wide tools and notations for modeling AK should be simple in order to lower the resistance by architects to learning a new tool. However, if such tools do not produce documentation up to company standards, they will be never successfully adopted by the organization.

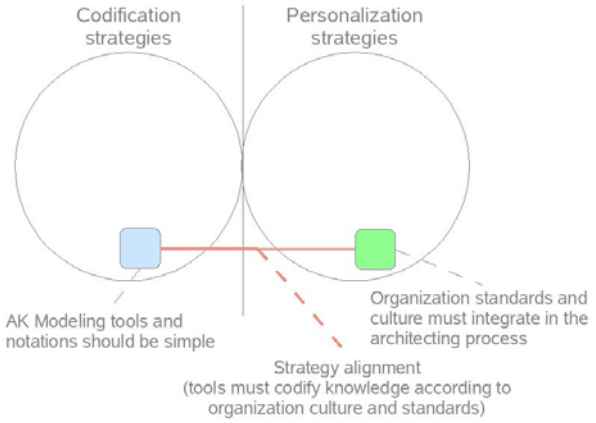

Figure 1 - Relation between codification and personalization strategies

The group was able to identify a list of best practices (or dos) and anti-practices (or don'ts) coming from the group's collective experience, as illustrated in Table 1. The group identified more codification than personalization strategies, reflecting the group's natural bias toward basic research in decision modeling.

We also looked at these codification and personalization strategies from the perspective of the goals a certain strategy addresses. We identified three relevant types of goals:

- Economic – having a positive or negative impact on the enterprise business;
- Social – solving or hindering organization processes, human interactions or playing a positive or negative influence on cultural aspects.
- Technological – providing a solution to an IT problem or representing an IT challenge.

We further tried to relate all strategies to the goals, as illustrated in Figure 2. Again, this mapping reflects the personal experience of the working group, together with the individual assumptions and interpretations about the most relevant goal addressed by a certain strategy. The goal can be visualized by the position of the small square in one of the six quarters. For example, consider strategy C-02 (defined in Table 1). Codification support should be tailored around the needs of the people consuming existing AK. Unfortunately, tool developers typically focus their efforts on the *producers* of AK and inadequately consider support for the *consumers* of AK. This strategy primarily addresses economic goals, because a company that invests in capitalizing its AK must adequately adopt this strategy or else resources and knowledge will be unused, resulting in economic losses. Moreover, there is also a positive impact towards social goals. With a strong focus on consuming AK, people will be provided with positive incentives to share their experience and to learn from one another, possibly leading to a high quality working environment.

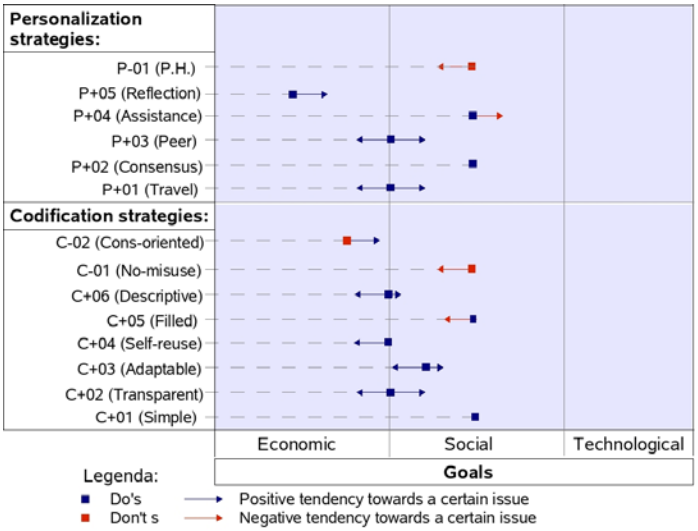

Figure 2 - Issues addressed by dos and don'ts

The visualization proposed by the working group needs further work, especially with respect to giving a more precise meaning to the positioning and the tendencies. Even with its shortcomings, this graph may help illustrate the evolution of strategies over time within an organization.

From this visualization we can further observe that most maturity seems to happen in the social area (12 out of 14 strategies). A possible reason for this bias could be that researchers in computer and information sciences are traditionally much more skilled in codification than in personalization. This reflects a natural bias towards the technological aspects of implementing a system than in the process and management aspects of investigating the industrial needs and identifying IT support.

Similarly, the visualization also shows that the group could not center any of the listed strategies with technological goals *per se*. The group was convinced that technological goals were important, but it is possible that they simply consider technology as a means to solve a goal, and not as a goal itself.

In one way or another, the working group gained more insight about codification strategies than the role of socialization in architecting. Even though the identified strategies come from industrial practice, a convincing analysis of their economic impact is needed for companies to adopt architectural knowledge management strategies. More quantitative research in collaboration with industry is required to perform this analysis.

Codification		Personalization	
		DOs	
C+01	**Simple.** Tools and notations should be simple, in order to create incentives and lower the learning curve.	**P+01**	**Travel.** The traveling architect carries AK across borders and enables sharing.
C+02	**Transparent.** Tools should be easily usable during the process (and not after its completion) to enable seamless integration.	**P+02**	**Consensus.** Practices aimed at team building (e.g. peer work) ease AK cross-fertilizations and create incentives.
C+03	**Adaptable.** Tools (templates) should be flexibly customizable, to accommodate a different practice.	**P+03**	**Peer.** (Best) practices come from peers (top-down imposition of compliance rules should be avoided).
C+04	**Self-reuse.** Codification must be useful for yourself, not for others (to create incentives, remove the old idea of creating a knowledge base that will 'replace people', and support training of new employees).	**P+04**	**Assistance.** Use techniques to assist people in their work, like history-based automation (e.g. automated tailoring like Amazon's previous history; feedback about something done previously).
C+05	**Filled.** Start a strategy with a filled knowledge base (a 'starting set' of useful AK).	**P+05**	**Reflection.** Time for after-the-fact reflection should be allocated explicitly in project management.
C+06	**Descriptive.** Tools should support the user in the decision making process (descriptive) rather than impose a solution (prescriptive).		
		DON'Ts	
C-01	**No-misuse.** A codification approach should clearly support positive goals and NOT be misused to penalize people.	**P-01**	**PH (Pigeon hole).** Skills and responsibilities are grouped in clearly separated compartments (pigeonholes). People covering them run the risk to remain 'blocked' with a role.
C-02	**Cons-oriented.** Effectiveness in the consuming activities is as important as in the producing ones.		

On the link between codification and personalization

DOs

CP+01	Tools supporting codification should support incremental community building.
CP+02	Alignment between codified models and organization documentation standards is mandatory.
CP+03	Organization culture must keep aligned codification and personalization.

Table 1 – AK Dos and Don'ts

Adopting an AK-centric approach in an organization

An AK-centric approach to software or systems engineering is one which emphasizes the importance of explicit documentation and reuse of Architectural Knowledge. As illustrated above, there are many social, economic, and technical factors in the acceptance of an AK-centric approach. An organization investigating the adoption of an AK-centric approach needs first to consider the people involved and how AK can help them in achieving their goals. A working group on process adoption within the organization was formed to discuss these issues.

The discussion started by asking, in the terms of IEEE 1471, "who are the stakeholders with AK-related concerns?" The group considered 8 different types of stakeholders as most significant: Architects, Maintainers, Customers, Managers, Process engineers, Knowledge engineers, Tool developers and the rest of the development team. The stakeholder category encompassing other members of the development team, such as the requirements engineers, or the testers, certainly have an interest in AK, but only a peripheral role in producing and consuming AK. The inclusion of a process engineer may be surprising, but during the discussion, managing AK within the process was identified as especially important. The group then discussed two of the stakeholders in more depth: the architects and the maintainers.

The architects are the main producers of AK, as they make most of the important decisions while architecting. The degree that architects produce explicit AK depends highly on how they view their own work, and this perspective varies widely from organization to organization. From our experience with industrial projects, architects tend to have a short-term, forward engineering view when architecting. Specifically, they are not particularly willing to invest time to produce AK that can be used later in the system's life cycle if they do not see a direct personal benefit. Architects often leave a project after the initial architecture design phase and have no further stakes in the project. Sometimes they are not the "owners" of design decisions and consequently are not responsible for them. This is true not just for the designated architects, but especially for consultants and subcontractors. Architects may plan for evolution but they do not evolve the system themselves; evolution is usually the job of the maintainers.

The maintainers (in the broad sense of the term) are the main consumers of AK that need to understand past design decisions and make new ones. The principle motivation for adopting an AK-centric approach comes from the need to understand the system during evolution. Therefore, maintainers have a critical need for AK, as they spend most of their time discovering knowledge about the design "after-the-fact". In this respect the research community must survey the real needs of maintainers in consuming AK.

One type of AK that is of particular importance to maintainers is the consideration of alternative or failed solutions that were considered or explored but rejected. Maintainers need to learn from past mistakes, understand the associated rationale and avoid repeating similar mistakes. Unfortunately alternative solutions from past decisions are not usually documented, even if exploring the rejected solution consumed time and resources. In practice, architects do not always explicitly reason about alternatives – *i.e.*, they unconsciously reduce the solution space.

If an organization aims to adopt an AK-centric approach, it needs to define its own use cases for AK and incorporate them into its own processes. However, in practice, even when processes explicitly make provisions for documenting AK, there never seems to be enough time to document AK with enough rigor to be useful. AK, even if well documented in the first place, will eventually become out of date if not maintained. A major problem is that documenting AK is traditionally considered an activity additional or supplementary to architecting. Typically, a scribe documents what the architect has decided after-the-fact. The architecting processes need to be enhanced with lightweight yet effective ways of AK documentation. Documentation of design decisions should be a by-product of architecting, so that the creation of explicit design models and design decisions are an inexpensive benefit of the use of an architecture design technique.

Tooling can support different uses of AK, and the working group was primarily interested in searching mechanisms, both in informal, unstructured data and codified AK (*e.g.*, in a knowledge base). There is currently much AK-relevant information hidden in documentation, but it is difficult and expensive to codify it. Stakeholders waste time and resources trying to locate the appropriate information using naïve free text searches, and valuable AK is not reused because it cannot be identified. If AK can be formally or semi-formally encoded, more precise queries can be used, and knowledge bases from different organizations can be combined. The challenge in this case is to derive appropriate ontologies to formalize AK, and to translate domain- and organization-specific ontologies across projects and organizations.

Another important type of tooling is the one that supports the capture of AK. The group envisioned a tool that can be customized to the architecting process and asks questions at the right times. For example when an architect commits a new decision, the tool could interact with the architect in order to capture the rationale. A system such as this should be descriptive and not prescriptive. Architects do not want to have structures and decisions imposed on them but rather make the decisions themselves and have the tool perform routine automation and provide relevant views on supporting information. It should also aim at reusing as much application-generic AK [7] as possible, both from the problem space (*e.g.*, reusable constraints) and from the solution space (*e.g.*, patterns, styles, and tactics.)

In an organization, the AK may be shared by different stakeholders, depending on communication issues and political processes. It is important to emphasize that adopting an AK-centric in an organization does not mean all design knowledge becomes public and is shared indiscriminately. Some AK needs to be kept secret to preserve confidentiality; for legal and business reasons; and because of social, cultural and political aspects.

In summary, the working group identified certain steps an organization can take in order to adopt an AK-centric approach. First, producing and consuming AK should not be considered an extra, resource-consuming activity but it should be made "invisible" – part of the organizational process. Second, people need to be convinced of the long-term benefits of such an approach in order to establish the corresponding culture. The role of the customer must be emphasized here. The customer needs to be convinced that investment in documenting AK is worth the extra budget. Third, the management of organizations should impose the necessary processes from top to bottom - especially in big organizations, people will follow rules. Finally, the working group emphatically declared the need for more empirical evaluation of AK-centric approaches in organizations. Success stories with verifiable claims can help encourage organizations to adopt AK-centric approaches, and negative studies can help guide research in evolving AK practices to better meet the needs of organizations designing and evolving complex software-intensive systems.

Tool support for AK

The working group on tool support focused on the issues involved with capturing, using, locating and maintaining AK. The attendees talked about their experiences in developing tool support and attempted to identify areas for improvement in the current state of the practice with respect to tool support for AK.

The most important attributes of an AK tool are what knowledge is explicitly modeled in the tool and how the tool is intended to be used in the context of a development process. The design of most AK tools can be classified as derived from data modeling or derived from process analysis. In tools derived from data modeling, there is a tendency to capture everything that is available at some time on the assumption that it is cheaper to capture data during design than to attempt to reconstruct it later. However, if there is not a clear idea of how data will eventually be consumed, the effort of modeling and capturing it is wasted. Process analysis is important in designing tool support, but it is always the case that a tool itself introduces changes to the process. Care must also be taken that a process-derived tool can be personalized sufficiently to make it useful across many projects and organizations.

The general consensus is that the primary capacity of tools should be to provide support to software architects. The information captured within a tool should be available to a human architect in such a way that it facilitates decision-making, but that the tool itself should not be prescriptive, or even advisory, in its capacity to support the architecting process.

AK tool support should integrate with existing process and tooling. Workshop participants reported success with tools based on Web interfaces, integration with UML-based modeling environments, and even standard word processor and spreadsheet applications using macros. Integrating with existing tools and processes makes the tool simpler and more intuitive to use.

The current state of research has been focusing on capturing data using AK tools, but issues in locating and consuming data within AK tools is still a very open issue. Two approaches to formulating queries into AK data are free text searches and first-order predicates. Unfortunately, free text searches are unstructured and difficult to identify relevance. First-order knowledge does not scale well over large data sets. It is increasingly difficult to identify what is present in AK databases and how it can be consumed. There has been little empirical analysis on how AK is used by consumers, what query patterns are most useful, and how to identify relevance of query results.

When it comes to maintaining evolving AK, the most important factor is traceability. Associations between AK are lost over time, and as the data set increases, managing those associations and dependencies becomes more difficult. Some tools already have basic decision dependency and traceability support [4].

Measurement and Evaluation of AK

The working group on measurement and evaluation identified two major themes:

- Where in the organization to conduct measurement and evaluation, and
- How to carry out the measurement and evaluation study.

The generally obvious places were during initial architecture derivation and architecture evolution. . In addition, one of the more interesting suggestions was to measure and evaluate how AK is used in mentoring, specifically, to understand what in an architect's experience in terms of AK needs to be passed on to those being mentored. Another situation where the applicability and relevance of AK should be studied is where a product-line architecture is being created out of individual architectures.

The discussion on empirical measurement design resolved into two subtopics: what should be measured and how to conduct the study. Some of the suggestions on what to measure include such basic quantitative measure as costs and resource utilization, and also qualitative properties such as ease and gracefulness of changes, patterns and strategies used, and utilization of knowledge reuse.

The discussion of study constructs and techniques included a number of interesting suggestions. One is the use of self-reports – keeping a log of what and how AK is used. Another is to look at what the consumers of AK need and use – that is, observation or ethnographic studies. Other approaches include case studies that include observation, interviews, and participation – all providing different viewpoints and different kinds of data. Controlled studies are important, but require significant resources in terms of time and people. The group recognized the need to non-intrusively study AK in the field.

Some of the problems encountered in terms of evaluation of costs are the usual ones: costs are paid up front and benefits are enjoyed later. Given the lifetime of a software architecture and its system, this imbalance poses significant challenges. However, it was noted that often managers may receive an immediate benefit from cost analysis studies. An important issue in interviews is the fact that seldom are the full data made available; it is usually only the distilled results that are available. This poses problems for reproducibility and meta-analysis (where we combine the results of several studies). Independence of replications is also a critical issue that is seldom dealt with.

Architectural decisions and rationale in IEEE 1471

Rich Hilliard invited SHARK/ADI workshop participants to provide input for the revision of the standard IEEE-Std-1471-2000 "Recommended Practice for Architectural Description of Software-Intensive Systems" (now also ISO/IEC 42010) [3].

The discussion focused mostly on how to expand the conceptual model of 1471 (Figure 1 of [3]), to define additional concepts, and to elaborate how these concepts are used in practice.

- A *Decision* is a choice of an element, property, or process that addresses one or more concerns, and affects directly or indirectly the architecture.
- A *Rationale* is an explanation – a justification associated with a Decision – which explains the reasoning (pros and cons, trade-offs, *etc.*) and points to the sources of knowledge.

A decision may address more than one concern. A decision may affect the architecture in several ways. A decision may:

- Specify existence of a architectural element,
- Constrain the property of some architectural element,
- Provide a trace between architectural elements and concerns, or
- *Raise* additional concerns.

Decisions are interrelated, and there are many different types of dependencies between decisions. These dependencies include constrains, enables, subsumes, conflicts, and others. A more complete list can be found in [6].

A design decision in particular is a ternary relationship associating: {concern, model, rationale}. We noted that there are architectural decisions related to the problem space, and not solely on the solution space. For example, prioritization of concerns is an architectural decision – though not strictly speaking a *design* decision.

In effect, the rationale for the architecture is made of the collection of all the rationale (instances) for all the decisions (instances), but for the same reason the conceptual model in the standard does not decompose a *model* into its constituent elements. Rationale is associated to model (or architecture) only indirectly through the decision.

Issues, options, decisions and rationale could be packaged by views and viewpoints, but workshop participants found numerous cases of concerns, issues, options, decisions and rationale that are not limited to one view and therefore these concepts should not be tied to views, which does not preclude an application of the standard to do so.

From a more detailed process perspective, a decision should address a clearly defined *Issue*, which is in turn associated with one or more concerns. An issue is a kind of concern that must be addressed by a decision. It provides the question, the problem statement. Figure 3 represents a complete view of the conceptual model.

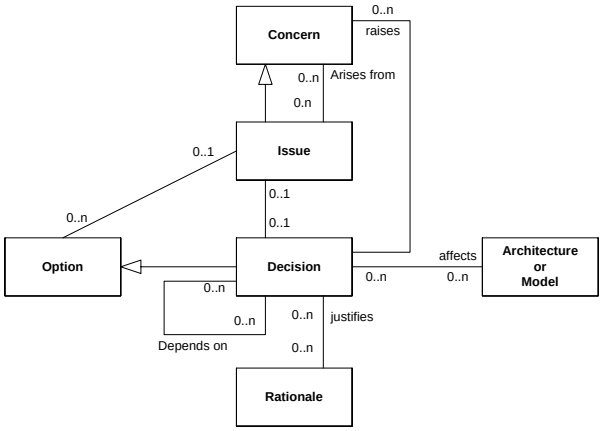

Figure 3 - Conceptual Model of a Design Decision

More work is required to better define what concerns and issues really are with respect to IEEE 1471.

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:

- Martin Becker, Fraunhofer IESE, Germany
- Jan Bosch, Intuit, USA
- Janet Burge, Miami University, USA
- Jeff Conklin, CogNexus Institute
- Rafael Capilla, Universidad Rey Juan Carlos, Spain
- Torgeir Dingsoyr, Sintef, Trondheim, Norway
- Muhammad Ali Babar, National ICT Australia
- Mike Evangelist, The University of Texas at Austin
- Paul S Grisham, The University of Texas at Austin
- Jim Herbsleb, Carnegie Mellon University
- Ralph Johnson, University of Illinois at Urbana-Champaign, USA
- Axel van Lamsweerde, Universite Catholique de Louvain
- Nenad Medvidovic, University of Southern California
- Dewayne E. Perry, The University of Texas at Austin
- Antony Tang, Swinburne University of Technology, Australia
- Jeff Tyree, CapitalOne, Canada
- Hans van Vliet, VU University Amsterdam, Netherlands
- Uwe Zdun, Technical University of Vienna, Austria

References

[1] J. C. Dueñas and R. Capilla, The Decision View of Software Architecture. In Proc. of the 2nd European Workshop on Software Architecture (EWSA), Pisa, Italy, 2005.

[2] R. Farenhorst, P. Lago, H. van Vliet. Prerequisites for Successful Architectural Knowledge Sharing. In Proc. of the 18th Australian Conference on Software Engineering (ASWEC), Melbourne, Australia, 2007, pp. 27-38.

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

[4] A. Jansen, J. van der Ven, P. Avgeriou and D.K. Hammer, Tool Support for Using Architectural Decisions, In Proc. of the 6th Working IEEE/IFIP Conference on Software Architecture (WICSA), Mumbai, India, Jan. 2007.

[5] P. Kruchten, An Ontology of Architectural Design Decisions, presented at 2nd Groningen Workshop on Software Variability Management, Groningen, NL, Rijksuniversiteit Groningen, 2004.

[6] P. Kruchten, P. Lago, and H. van Vliet. Building up and reasoning about architectural knowledge. In Proceedings of the Second International Conference on the Quality if Software Architectures (QoSA 2006), June 2006.

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

[8] J. Tyree and A. Akerman, Architecture Decisions: Demystifying Architecture, IEEE Software, Vol. 22, 2005, pp. 19-27.