Processes for Creating and Exploiting Architectural Design Decisions

Francisco Nava[1], Rafael Capilla[1], Juan C. Dueñas[2]

[1] Department of Computer Science, Universidad Rey Juan Carlos,
c/ Tulipán s/n, 28933, Madrid, Spain
{francisco.nava, rafael.capilla}@urjc.es
[2] Department of Engineering of Telematic Systems, ETSI Telecomunicación
Ciudad Universitaria s/n, 28040, Madrid, Spain
jcduenas@dit.upm.es

Abstract. The software architecture research community has recently stated that software architectures suffer a serious lack of documented design decisions as well as other relevant architectural knowledge. Architecting processes usually don't record the decisions that led to the final architecture, which become very important to reduce the effort spent in maintenance and evolution activities. In addition to the variety architectural knowledge items that can be stored during any development or maintenance activity, several processes and complementary activities can be define to store, manage and reuse such architectural knowledge. This paper deals with the processes hat become relevant to store and manage architectural design decisions. In addition, we provide some preliminary results of the experience carried out with ADDSS, a web-base tool aimed for recording, managing and documenting design decisions.

1. Introduction

The popularity of software architectures as the central cornerstone for building software systems has attracted the attention of many researchers from the software engineering field. The development of software systems involves several stakeholders with different requirements that may contribute to the success of the architecture. Architectures [1] [17] can be described by means of different architectural views [10] [16] that reflect the interests of the stakeholders. In addition to this, both the intermediate products as well as the final architecture, all of them generated during the design phase, are build as a result of a set of design decisions made by the architect [2]. From the fact that software changes and evolves over time, most software engineering activities are immersed in a decision making process that tries to meet the goals specified in the requirements.

To date, most researches have focused their attention to architecture as a "product", which have been documented in the form of several architectural views. A step beyond deals with "architectural knowledge (AK) as a product" [12], which considers architectural design decisions, meta-models, and notations for documenting architectural knowledge which has not been explicitly recorded as way to avoid knowledge vaporization. During the last three years, the software architecture community has stated the need to record, manage, and document explicitly the rationale and the design decisions that make possible to create our software architectures. The expectations from the benefits of codifying and recording this architectural knowledge should not be only focused on design decisions as themselves. Moreover, the importance for recording design decisions has been recognized as a way to bridge the gap between requirements and architectural products and for traceability issues as well. Furthermore, other complementary knowledge related known as “architectural knowledge as a process” [12] becomes quite important for development and maintenance process. Those activities concerning to architecting, sharing, reuse, and discovering architectural knowledge have a strong impact when we build or modify the architecture.

In this paper we will focus of the activities belonging to “architecture as a process” and how some of these have been used in ADDSS, a web-based tool for recording and managing design decisions. In addition, we present some preliminary result of using ADDSS. Finally, we provide the conclusions and the future work.

2. Design Decisions in Software Architecture

Just early in the ‘90s, Perry and Wolf [14] mentioned the rationale and principles that guide the design and evolution of software architectures. This rationale has to be taken into account when adopting an architecture-centric approach to justify the form and the elements of the architecture. An updated of these ideas are reflected in [5] which describes that design decisions have to be documented explicitly, but not the processes that lead to them.

Different authors have addressed in recent works the problem to reflect architectural design decisions as part of the architecture documentation. For instance, Tyree and akerman [19] provide a template for charactering architectural knowledge but they pay little attention to the processes to create and manage such knowledge. In [18], the authors motivate the need for documenting architectural design decisions. Recording architectural descriptions mainly based on diagrams describing the relationships between components and connectors seems to be not enough. Many maintenance processes become highly-cost processes because of architecture erosion or non existing designs (that may lead to an architecture recovery process). The lack of design decisions previously recorded and the rationale that motivated them hampers the maintenance activities and makes difficult the process to replay the decisions taken. Other authors [15] focus on the explicit representation of assumptions for architectural design decisions as way to make explicit tacit knowledge which is implicit in the architect’s mind. In summary, the characterization of the rationale used in the construction of software architectures must be explicitly documented, as a complementary view which crosscuts the traditional architectural views [7].

The approaches of other authors describe a meta-model to support the characterization of architectural design decisions but linking them to architectural products. For instance, the architecture-centric concern analysis (ACCA) method [20] employs a meta-model to capture architectural design decisions by linking them to software requirements and architectural concerns. In [9], the aim of the Archium approach is to support design decisions as first class entities. Archium defines a meta-model which is composed of three sub-models: an architectural model, a design decision model, and a composition model which compose design fragments (an architectural fragment defining a collection of architectural entities). Archium is also a component language which extends Java for describing components, connectors and design decisions with tool support. In [4], the authors characterize architectural design decisions using a meta-model and a web-based tool called ADDSS (Architecture Design Decision Support System, http://triana.escet.urjc.es/ADDSS) which focuses on the processes for recording, storing, managing and documenting design decisions following an iterative approach. Both Archium and ADDSS approaches mention some of the processes that should be carried out to create and use architectural design decisions, but a more detailed description is needed.

3. Processes for Architecting with Design Rationale

As discussed in [12], architectural knowledge as a process "deals with using architectural knowledge during the software development lifecycle". Use cases, methods for recording and discovering knowledge, tools and services for supporting the use of AK fall on this category. The decision-making process performed during the construction of any software architecture may accomplish a certain number of activities associated to the creation and usage of AK. Also, there are a set of relationships between these activities and sub-activities that influences the way in which the architecture is created. Some activities may relate to other activities, some of them are implicit in the design phase while others have explicit relationships. All these activities can be associated to basic and complex use cases that can be classified into two main categories: the producer side in which knowledge is stored and the consumer side where the storage knowledge can be (re)used and discovered. [12]. Following this initial attempt to classify the activities that codify and use AK, we detail in this section other sub-activities and their relationships between them that are of common use during the architecting process. In addition to this, we define the logical order in which all these processes should take place. Some of the activities described here are mentioned in [12] and are marked with an "*". The refinement into sub-activities and the relationships among them belongs to this work.

(a) **Architecting***: Concerns to the process for creating and storing architectural knowledge (design decisions, rational, assumptions) as well as all the attributes that characterize this AK.

- **Analyze functional and non-functional requirements:** Before a decision is made, the architect has to identify which requirements will be part of the decision is being made.
- **Evaluation of already stored AK:** The rationale of a new decision can be based on already codified knowledge (e.g.: patters, styles) or it could be

based on previous decisions and past experiences. The architect's expertise heavily influences this activity.

    - **Knowledge search / discovery / reuse:** Knowledge search processes and discovery methods are particularly welcome [6] to extract useful knowledge.
        - **Document browsing:** In its most simple form, knowledge browsing can be used to search well-known patterns.
        - **Queries:** A more elaborated way constitutes the use of queries to identify and extract knowledge (e.g.: past decisions). Reuse of AK may include some way to discover and to identify which type of knowledge would be more suitable for the target decision.
        - **Reasoning processes:** Depending how AK is codified, processes for reasoning about AK can be set up (e.g.: for ontologies [11]).
- **Assessment for architecting:** Link to assessing activities.
    - **Store alternative decisions:** All the information concerning to the alternatives considered is recorded.
- **Make assumptions:** Assumptions can be made or not before a decision is made, but in many cases, recording explicitly assumptions are helpful to understand the rationale of the future decision. Assumptions can be also represented graphically in the architecture.
    - **Record assumption**
    - **Represent assumption**
- **Make decisions:** Finally a design decision is made based on the alternatives evaluated and on the assumptions made. Decisions are stored for future use and they can be also represented in the architecture.
    - **Record decision**
    - **Represent decision**
- **Characterize decisions:** In this sub-activity we can characterize the decisions made with some attributes, like for instance: the status (e.g.: pending, approved, obsolete), type of decision (e.g.: main, alternative), responsible, etc.
- **Establish dependencies:** After one or more initial decisions are made, it is possible to establish dependencies and links from new decisions to past ones. These dependencies will be motivated by requirement constraints. A link to constraint evaluation could be possible.
- **Build reusable AK:** This sub-activity is a complex task for which no guidelines are available. We can think of simple well structured AK documentation or they way in which decisions are stored for reuse (e.g.: by means of queries). This sub-activity is not mandatory for the architecting phase.
    - **Document AK:** Link to document AK.

(b) Sharing*: By knowledge sharing we mean making AK available for others [12]. Several ways can be used for transferring AK [8], but due to the difficulty to draw the boundaries between consumers and producers and because we need to determined which AK is worthy to be shared, we have classified sharing processes into two main ones.

- **Passive sharing:** The architect reviews existing AK (e.g.: documentation, books, web pages, codified knowledge) and uses it for the architecting goal. We assume this existing knowledge have been documented to be shared by others.
    - **Document AK:** Documented AK becomes an implicit tasks that can be part of any architecting process for sharing purposes.
- **Active sharing:** In this case, meetings and interviews with other architects and stakeholders become a way for sharing knowledge.
    - **Pull / Push strategies:** Publisher subscriber strategies or mechanisms can be used to publish relevant information and subscribe to it (e.g.: RSS contents for distributed teams).
    - **Knowledge search / discovery /reuse:** Link to Knowledge search / discovery /reuse.

(c) **Assessing*:** Includes those processes that evaluate and assess the architect about viable decisions as well as the better and worst ones taking into account the existing information, constraints and requirements. The expertise from the architect and other relevant stakeholder are used for assessing activities. We can have pre and post assessment activities, such as the following ones.

- **Assessment for architecting:** Using the previous knowledge, the assessment will come on to decide which will be the best decision. Alternatives are usually considered as well as the pros and cons for each alternative. The problem arises because often, alternatives are only in the architect's mind.
    - **Alternative evaluation:** Alternatives can be evaluated and stored for future maintenance purposes.
    - **Evaluate pros / cons:** For each potential alternative, the pros and cons must be recorded and afterwards, analyzed as part of the decision making process.
    - **Evaluate quality attributes:** In those cases where is needed, an evaluation of quality aspects [13] for several candidate architectures or for specific pieces or software-hardware pieces of the future system may determine the final decision.
    - **Constraint evaluation:** In many cases, restrictions specified in the requirements limit the solution space for a particular alternative. In other cases, one decision depends from another already made.
- **Assessment for learning:** Some assessment procedures after the architecting phase is finished can be performed to explain to the stakeholders how a particular architecture is built. Assessment for learning activities can be performed during the life of the system an in particular when maintenance operations are carried out. In this case, we can teach and learn about past wrong and right decisions made. New developments can benefit from past experiences.

(d) **Learning*:** Users can learn form past decisions, both from right and wrong ones. Learning from previous experiences influences positively the construction of new architectures and leverages the expertise of non-expert architects and beginners. For maintenance processes, learning from past decisions, or following traces between requirements, architectures and decisions, constitute a valuable knowledge base from which maintenance can be benefit. We think in different learning techniques.

- **Navigation / browsing:** Users can navigate through the decisions made, and browse decisions and other codified AK. In addition, Internet search can be used to discover documented experiences.
- **Knowledge search / discovery / reuse:** Link to Knowledge search / discovery /reuse.
- **Training**
- **Assessment for learning:** Link to assessment for learning.

We have used mindmaps (http://en.wikipedia.org/wiki/Mind_map) as a diagram to represent the ideas and the tasks described in this section. Mindmaps constitute an expressive way to describe ideas around a central word or idea which can be clearly visualized and structured as well as an aid in decision making. Figure 1 describes the mindmap of the tasks already mentioned. A uses green arrow represent the links between certain tasks. Also, architecting and sharing phases are coloured in green the highlight the "producer side" while assessment and learning are in orange because they belong to the "consumer side."

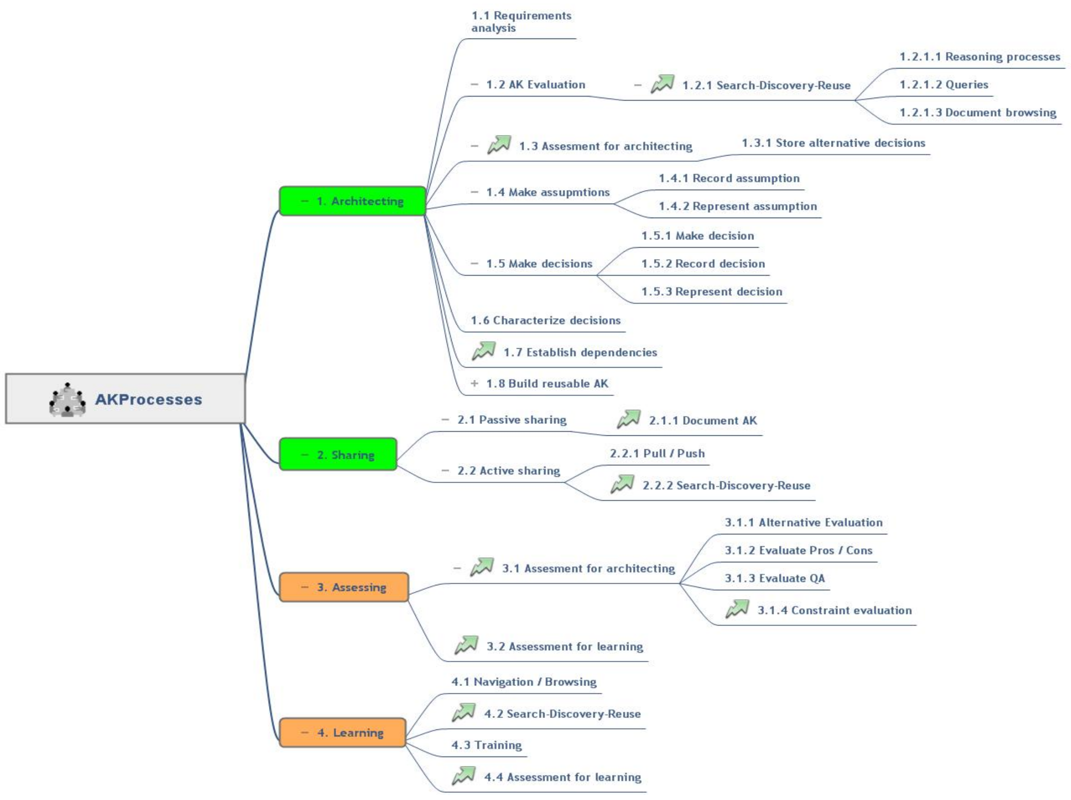
AKProcesses
1. Architecting
1.1 Requirements analysis
1.2 AK Evaluation
1.2.1 Search-Discovery-Reuse
1.2.1.1 Reasoning processes
1.2.1.2 Queries
1.2.1.3 Document browsing
1.3 Assesment for architecting
1.3.1 Store alternative decisions
1.4 Make assupmtions
1.4.1 Record assumption
1.4.2 Represent assumption
1.5 Make decisions
1.5.1 Make decision
1.5.2 Record decision
1.5.3 Represent decision
1.6 Characterize decisions
1.7 Establish dependencies
1.8 Build reusable AK
2. Sharing
2.1 Passive sharing
2.1.1 Document AK
2.2 Active sharing
2.2.1 Pull / Push
2.2.2 Search-Discovery-Reuse
3. Assessing
3.1 Assesment for architecting
3.1.1 Alternative Evaluation
3.1.2 Evaluate Pros / Cons
3.1.3 Evaluate QA
3.1.4 Constraint evaluation
3.2 Assessment for learning
4. Learning
4.1 Navigation / Browsing
4.2 Search-Discovery-Reuse
4.3 Training
4.4 Assessment for learning

Figure 1. Mindmap of activities and processes for building and exploiting architectural knowledge.

4. Recording and Documenting Design Decisions with ADDSS

In a prior work [4] we described ADDSS, a web-based tool for recording, managing and documenting architectural design decisions, as well as a list of requirements that should be considered to build a tool that supports the storage and use and use of architectural design decisions. With ADDSS, architectural design decisions are made following an iterative approach and one or more decisions can be made for each project's iteration. For each set of decisions, an image of the architecture can be uploaded to the system. Thumbnail images of the original architectures facilitate the visualization for all the iterations made. For each decision a set of functional and non-functional requirements (FR and NFR) can be associated, but ADDSS 1.0 doesn't distinguish explicitly between FR and NFR. ADDSS 1.0 doesn't support explicitly alternative decisions, but this problem can be overcome by adding all the alternative design decisions and after deliberation activities, select the right ones. Also, basic dependencies can be established between design decisions as a way to represent very simple constraints. At the end, the documentation of the project in the form of PDF documents can be generated automatically

We tried the tool in two small-medium projects but in this work we will explain the issues that arose during an evaluation carried out in a master course during the first semester of 2006-2007. Twenty-two master students participated in the evaluation using the tool organized in 11 teams of two persons. Fifteen out of 22 students work for a company while the other 7 are undergraduate students in computer science from the last course. In addition, 3 students are associate professors in the Telecommunication school of the Politechnic University of Madrid (UPM). The students were familiarized with software architecture concepts and architectural design decisions before using ADDSS, and all of them employed the same system that had to model using ADDSS.

The target architecture belongs to an immersive virtual reality (VR) system, which consists of a virtual church with a virtual guide tour inside the church. Before starting the architecture construction process, we populated the database with 23 design patterns and 4 architectural styles. A specification requirements list with 28 functional requirements and 5 non-functional requirements were given to the teams. We omitted other specific application requirements as well as hardware requirements. All the teams had around 20 days to make the design decisions according to the requirements list and using ADDSS in order to produce a final architecture. Intermediate architectural products and the final architectures were designed with Rational Rose or Magicdraw and uploaded to the ADDSS database during the decision making process. UML class diagrams belonging to the static view of the architecture were required because ADDSS 1.0 doesn't' support multiple views for a single project. Once the teams finished the evaluation of the tool, an application form was filled in order to extract the conclusions of their work and important feedback.

As a brief summary, some the conclusions we extracted after carrying interviews with the team's members at the end of the experience were the following:

Issue 1: The tool removes the requirements after a decision is made.

Comments to issue 1: ADDSS 1.0 was initial thought to support functional requirements but there is no restriction to upload NFR. Requirements deleted after each decision made is not a bug, because for FR we envision that one or more FR meets one or more decisions which implements a specific functionality of the system that will not appear in a different module. This is not the case of NFR for which one or more NFR may affect to one or more architectural pieces and software modules. Therefore, for the case of NFR, these must not be removed after each decision made.

Issue 2: The usability of the tool is high and easy to use, but the teams spend most of the time thinking about the decisions to be made and drawing the architecture. They feel the tool useful to record the decision made because they can work in parallel with architecture modeling tools.

Comments to issue 2: None.

Issue 3: Most of the teams employ between 4 and 6 project iterations designing the architecture, except two teams that employ more than 6 iterations.

Comments to issue 3: The longer experience of the members of the teams that employ more than 6 iterations lead to a more accurate design which provided detailed class diagrams. Otherwise, the number of decisions for each project's iteration was smaller compared to other teams. We can infer that teams are not familiarized to perform complex decisions or take several decisions for a single iteration.

Issue 4: Some of the decisions are not reflected in the architectures of subsequent iterations.

Comments to issue 4: This was a comment problem for all the teams. Apparently, most of the teams forget some decisions made and they don't reflect these in the refined versions of the architecture in subsequent iteration. For instance, if a package organization is selected for iteration 1, this package decomposition doesn't appear when detailing the classes of each package on iteration 2.

Issue 5: Teams perceive the usefulness of the tool for recording decisions, not only for development purposes but also for maintenance activities.

Comments to issue 5: None.

The results of the evaluation are shown in figures 2a, 2b and 2cb. Figure 2a shows the answers of each team for 6 general questions ranging from 1 to 10.

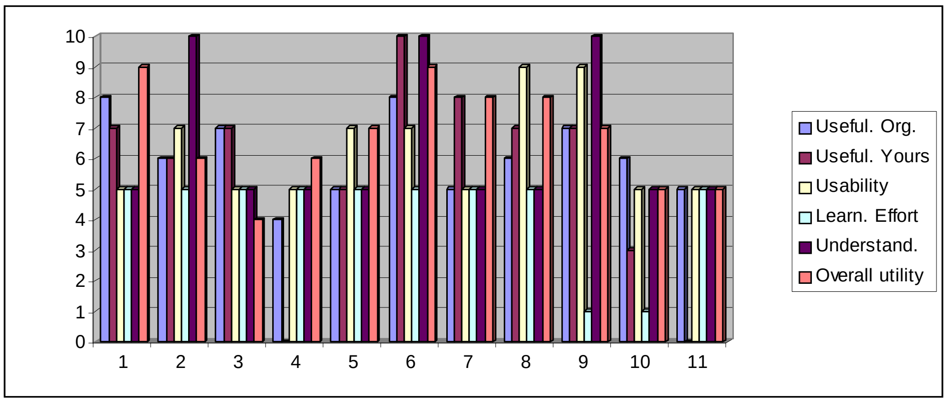

Figure 2a. Results of the evaluation of ADDSS. General questions

Figure 2b shows the hours spent by the all the teams for evaluating the ADDSS tool. Notice that 4 teams spent around 20 hours while 3 teams spent between 7 and 10 hours and 4 teams took less than 7 hours as average evaluating the tool. The reason might come from the expertise of the team members in their professional carriers.

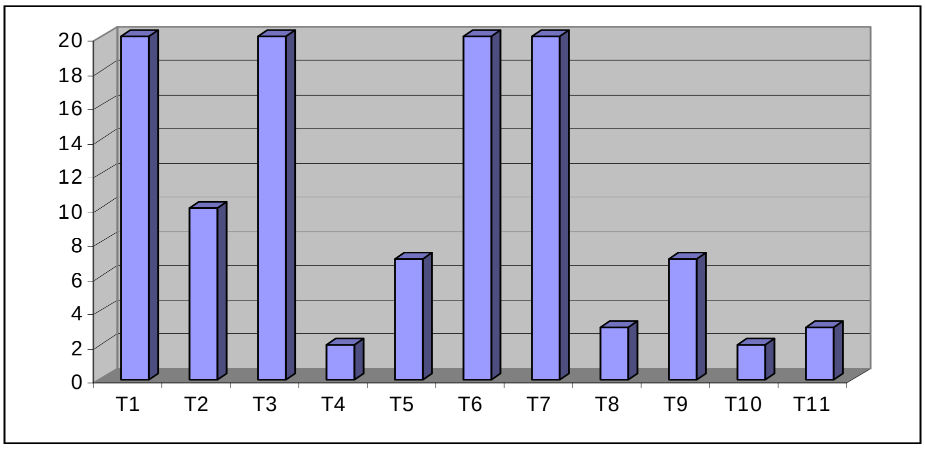

Figure 2b. Effort spent by the teams evaluating ADDSS

In figure 2c we represent the average of the values of figures 2a and 2b for all the teams. Only the usefulness of the tool for the team members is a little lower that the medium, while the rest of the values are over the average. Notice that value corresponding to the learning effort should be as lower as possible. Finally, average time of the evaluation spent by the teams is around 10 hours.

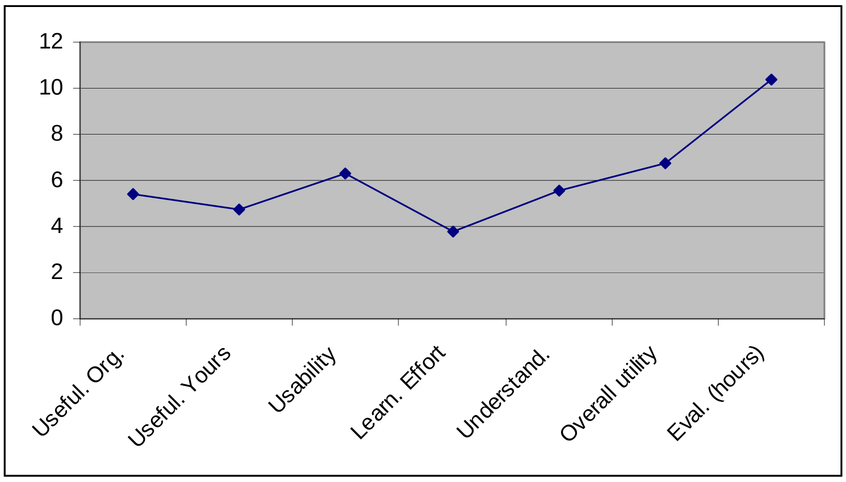

Figure 2c. Average values of the evaluation of ADDSS. General questions

6. Conclusions and Future work

Despite of some important missing features not implemented in ADDSS 1.0, the teams who evaluated the tool perceived the usefulness for recording and using architectural design decisions with tool support.

The current version of ADDSS (1.0) doesn't support all the tasks and features described in the mindmap of figure 1, but the tool is rapidly moving to version 2.0, which will support new important features not implemented for version 1.0.

References

1. Bass L., Clements P. and Kazman R. Software Architecture in Practice, Addison-Wesley, 2nd edition, (2003).
2. Bosch, J. Software Architecture: The Next Step, Proceedings of the 1st European Workshop on Software Architecture (EWSA 2004), Springer-Verlag, LNCS 3047, pp. 194-199 (2004).
3. Clements P., Bachman F., Bass, L., Garlan D., Ivers J., Little R., Nord R. and Stafford J. Documenting Software Architectures. Views and Beyond, Addison-Wesley, (2003).
4. Capilla, R., Nava, F., Pérez, S. and Dueñas, J.C. A Web-based Tool for Managing Architectural Design Decisions, Proceedings of the Workshop on Sharing and Reusing Architectural Knowledge, ACM Digital Library, Software Engineering Notes 31 (5).
5. Clements, P., Bachman, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R. and Stafford, J. Documenting Software Architectures. Views and Beyond, Addison-Wesley (2003).
6. de Boer, R. C. Architectural Knowledge Discovery, Why and How?, Proceedings of the Workshop on Sharing and Reusing Architectural Knowledge, ACM Digital Library, Software Engineering Notes 31 (5).
7. Dueñas, J.C. and Capilla, R. The Decision View of Software Architecture, Proceedings of the 2nd European Workshop on Software Architecture (EWSA 2005), Springer-Verlag, LNCS 3047, pp. 222-230 (2005).
8. Farenhorst, R., Tailoring Knowledge Sharing to the Architecting Process, Proceedings of the Workshop on Sharing and Reusing Architectural Knowledge, ACM Digital Library, Software Engineering Notes 31 (5).
9. Jansen, A. and Bosch, J. Software Architecture as a Set of Architectural Design Decisions, 5th IEEE/IFIP Working Conference on Software Architecture, pp. 109-118, (2005).
10. Kruchten P. Architectural Blueprints. The "4+1" View Model of Software Architecture, IEEE Software 12 (6), pp.42-50 (1995).
11. Kruchten, P., Lago, P., van Vliet, H. and Wolf, T. Building up and Exploiting Architectural Knowledge, 5th IEEE/IFIP Working Conference on Software Architecture, (2005).
12. Lago, P. and Avgeriou, P. First Workshop on Sharing and Reusing Architectural Knowledge, ACM SIGSOFT Software Engineering Notes, 3(5), 32-36.
13. Liao, L. From Requirements to Architecture: The State of the Art in Software Architecture Design, http://www.cs.washington.edu/homes/liaolin/Courses/architecture02.pdf
14. Perry, D.E. and Wolf, A.L. "Foundations for the Study of Software Architecture", Software Engineering Notes, ACM SIGSOFT, October 1992, pp. 40-52.
15. Roeller, R., Lago, P., van Vliet, H., 2006. Recovering Architectural Assumptions. The Journal of Systems and Software 79, 552-573.
16. Rozanski, N. and Woods. E. Software Systems Architecture: Working with Stakeholders Using viewpoints and Perspectives, Addison-Wesley (2005).
17. Shaw, M. and Garlan, D. Software Architecture, Prentice Hall (1996).

18. Tang, A., Babar, M.A., Gorton, I. and Han, J.A. A Survey of the Use and Documentation of Architecture Design Rationale, 5th IEEE/IFIP Working Conference on Software Architecture, (2005).
19. Tyree, J. and Akerman, A. Architecture Decisions: Demystifying Architecture. IEEE Software, vol. 22, no 2, pp. 19-27, (2005).
20. Wang, A., Sherdil, K. and Madhavji, N.H. ACCA: An Architecture-centric Concern Analysis Method, 5th IEEE/IFIP Working Conference on Software Architecture, (2005).