Extending Software Architecting Processes with Decision-making Activities[1]

Rafael Capilla, Francisco Nava

Department of Computer Science, Universidad Rey Juan Carlos,
c/ Tulipán s/n, 28933, Madrid, Spain
{rafael.capilla, francisco.nava}@urjc.es

Abstract. The traditional perspective on software architecture has paid much attention to architecting as a development process aimed at creating the architecture of a software system, as well as the documentation used to communicate the architecture to the stakeholders by means of several architectural views. Recently, the software architecture research community has faced the need to record, manage, and document the design decisions and the rationale that lead to such architecture. Because architectures are the result of a set of design decisions, this design rationale must be properly recorded and managed as a complementary process to the modelling activity. In this paper we detail different types of decision-making activities aimed at creating and using design decisions and how these can be supported with tool support.

Keywords: Software architecture, Architecture design decisions, Architectural knowledge, Architecting activity, Maintenance, Evolution.

1. Introduction

Software architectures have been successfully used in the past decades as the central cornerstone for describing the main functional parts of a software system [2], and the interests of different stakeholders are usually represented in the architecture by means of different architectural views [12] [17]. The more traditional perspective on software architecture [2] has paid much attention to modeling and documenting tasks while they have neglected the rationale that led to such designs. Recently, this point of view is changing to include the creation and use of architectural knowledge (AK) as a first class entity that should be recorded. As all architectures are the result of a set of design decisions [3], the impact and benefits of recording this AK seems to be promising for maintenance and evolution activities. Hence, as software systems evolve, the decisions made during the life of the system should evolve accordingly to the changes performed on the system and to new customer needs. Therefore, a continuous decision-making process happens to meet the goals specified in the requirements.

[1] This work is partially funded by the PILOH project of the Spanish Ministry of Education and Research programme under grant number URJC-CM-2006-CET-0603.

Recently, the software architecture community has recognized the need to record, manage, and document explicitly the rationale that lead to the creation of any software architecture. Architecture design decisions become now more important as they bridge the gap between requirements and architectural products. Thus, also traceability in maintenance activities can benefit from this approach.

In this paper we focus on those processes needed to deal with design decisions as a complementary product of the architecting activity. Also, we describe how some of these processes are supported by ADDSS, a web-based tool for recording, managing, and documenting design decisions. The structure of the paper is as follows. Section 2 discusses the representation of design decisions in software architecture. Section 3 deals with the processes that affect the creation and use of AK. Section 4 describes which of the processes mentioned in Section 3 are supported in the ADDSS approach. Section 5 provides some conclusions and outlines possible future work.

2. Representing and Creating Architectural Design Decisions

In the early 90s, Perry and Wolf [15] mentioned the rationale and principles that guide the design and evolution of software architectures. This rationale is used in the reasoning activity as the underlying reasons that motivate the selection of a particular architecture. These ideas have been detailed in [6] to state the need for documenting explicitly architectural design decisions, but not the processes that lead to them. Nevertheless, prior to the definition of the activities that should take place in the creation of such architectural knowledge (AK), it seems necessary to know which kind of information we should represent as part of the design rationale. Design rationale is the justification behind decisions, and different authors have addressed the problem to reflect design decisions as part of the architecture documentation [8]. Tyree and Akerman [19] provide a template list of items for characterizing architectural design decisions. In [18] the authors mention the need for documenting design decisions, because documenting architectural descriptions often based on a component & connector view is not enough. One of the reasons to store this AK comes from the need to carry out highly-cost maintenance processes motivated by architecture erosion or from non existing designs because design decisions were never recorded. Others [16] focus on the explicit representation of assumptions as a way to make explicit the tacit knowledge which is often implicit in the architect's mind. In [5], the authors propose a list of attributes which classifies design decisions into mandatory and optional attributes that can be tailored for each particular organization, as well as a set of attributes specific for describing the evolution of architectures. A meta-model combines the characterization of design decisions with the processes used to manage such knowledge. Similarly, the architecture-centric concern analysis (ACCA) method [21] uses a meta-model to capture architectural design decisions and linking them to software requirements and architectural concerns. The approaches mentioned before highlight the relevance for characterizing the architectural knowledge, but the processes that lead to it are only slightly mentioned.

2.1 Lifecycle for AK Creation

In addition to the AK representation, creating and using AK has to be integrated under the "natural" lifecycle of the more traditional architecting and engineering activities. To date, most software architects have seen architectures as a "product" that has to be maintained and evolved as requirements change. According to [3] [14], architects are changing their more traditional perspective by considering *architectural knowledge as a product*, which should be seen as first class co-product of the architecting activity in order to avoid knowledge vaporization. In addition, *architectural knowledge as a process* [14] "deals with the processes that create and use such AK during the software development lifecycle". Use cases, methods for recording and discovering knowledge, tools and services for supporting the usage of AK fall on this category. In this new scenario, the stakeholders involved in the development of any software architecture may act as "producers" and "consumers" of this AK. According to the classification defined in [14], *architecting* and *sharing* activities belong to the producer side while *learning* and *assessment* belong to the consumer side. These activities have been roughly described in [14] but they need some refinement in order to understand the detailed processes concerning to the creation of AK. Our main contribution in this paper focuses on a more detailed list of the processes and sub-processes that happen during the decision-making activity, as a refinement of the main ones described in [14], such as we outline in next section.

3. Activities for Recording and Using Architectural Knowledge

The activities concerning with the creation of AK are described in table 1. AK. Hence, before a decision is made, a reasoning activity may take place [13]. This reasoning process is based on the rationale and the motivation that guides a decision. The rationale often relies on assumptions made as well as on the analysis of the pros and the cons (i.e.: the implications) of each particular decision. Moreover, we have to take into account the existence of constraints for the decisions as well as the dependencies that may appear between current and previous decisions. Once a decision is made, we should give a concrete status (e.g.: pending, approved, rejected, obsolete) and store it in a readable form for subsequent use. Often, before a choice is selected, several alternatives can be considered. The evaluation of these alternatives means to deal with new decisions and sometimes search for codified AK. In addition, evaluation and assessment activities may happen and used to evaluate between different candidate solutions. Also, depending on the specific phase or project milestone, not all the existing AK may be needed at the same during the decision making activity. For instance, we can store a minimum set of attributes to characterize a design decision during the initial development phase, but a subsequent testing or maintenance activity may need extra attributes (e.g.: responsible, status). In practice, as much of these attributes are stored during the creation of AK more comprehensible would be the decisions made. For each main category of the processes defined in [14] (marked with an asterisk in the tables) we have detailed the set of activities and sub-activities that we believe belong to each category.

Table 1. Activities for creating architectural design decisions

ARCHITECTING (*): Creates and stores AK		
Activity	Sub-activities level 1	Sub-activities level 1
Make decision	Reasoning (rationale, motivation) Select the best alternative	Make assumptions Analyze implications Constraint and dependency analysis Evaluate AK Validate before storing
Characterize decision	Assign status and other relevant items	
Store and document decisions		
Evaluate AK	Reuse AK Evaluate alternatives	Search, Discovery Assessment / Learn

Once an amount of decisions has been stored, this AK can be shared with others. The processes that fall in this category are defined in table 2. In many cases, the boundary between producers and consumers for sharing activities is not clear in many cases. Producers share available knowledge to other stakeholders. AK producers may act also as consumers of codified knowledge. Moreover, architects may share AK with other architects, all of them participating in the development process. For instance, during architecting a well-known pattern can be shared to other architects to discuss its applicability as a suitable design solution. In other cases, once a set of design decisions are made and the first version of the architecture is built, a subsequent maintenance process might need to share some of the decisions made with others interested in learning from previous experiences. From our point of view, knowledge sharing can be a more passive task when the stakeholders review existing AK or even when they query a knowledge base. A more pro-active approach can take place if we want to publish knowledge to others that act as subscribers of such AK (e.g.: use of RSS contents for distributed teams). Active publishing-subscribing strategies as well as discussion groups can provide a more dynamic usage of codified knowledge. Moreover, brainstorming meetings can be organized to share and communicate this knowledge. In this case, knowledge sharing requires the participation of at least two or more stakeholders to achieve the communication goal, while a review activity can be done by a single stakeholder that learns from available knowledge.

Table 2. Knowledge sharing activities

SHARING (*): Make AK available to others		
Activity	Sub-activities level 1	Sub-activities level 2
Review AK	Analyze documents or existing AK stored	Search, Discovery
Communicate AK	Subscribe to AK Organize meetings	Pull/ Push (RSS) Discuss / explain

Complementary to AK producers, knowledge consumers include assessing and learning activities, as shown in tables 3 and 4. Assessment provides the guidelines and recommendations for selecting the best or the optimal decisions among several. The expertise of the architects and the results from evaluating different alternatives usually drive these assessment activities. Table 3 shows different assessment activities and sub-activities to assess before or after decisions are made. Sometimes, assessing about decisions needs from a previous learning activity in order to perform the right assessment. In such scenario we could perform assessment during architecting to select the best decision or during a learning activity to teach about future decisions, as architects can learn from right and wrong experiences. Assessing about AK can be used to know the viability of future decisions and provide further recommendations.

Table 3. Assessment activities with architectural knowledge

ASSESSING (*): Recommends the selection of a decision		
Activity	Sub-activities level 1	Sub-activities level 2
Evaluate	Evaluate impact of implications Constraint analysis Evaluate impact of quality attributes	Analysis of alternatives Simulation Impact analysis
Review	Check for completeness and correctness of AK	
Validate	Check decisions against requirements and architectural products Check the integrity of the dependencies between decisions	Traceability
Recommend	Communicate to stakeholders the results of the assessment activity	

The last activity concerns to learning tasks. Architects become more expert consumers of AK as they learn from past experiences. Learning improves also the career of architects from beginners to more expert ones. As a result, future architecting activities are expected to be performed better that initially. As shown in table 4, some learning activities include the evaluation of stored AK as a way to learn which of the decisions made were right or wrong, or to detect inconsistencies in the decision model.

Table 4. Learning activities from previous architectural knowledge

LEARNING (*): Understand why decisions were made		
Activity	Sub-activities level 1	Sub-activities level 2
Evaluate stored AK	Compare the decisions to products and requirements Detect wrong decisions or inconsistent AK	Follow trace links Search-Reuse AK
Training	Teaching about past decisions and experiences	Search-Reuse AK Assessment / Learn

From our point of view, assessment and learning are often intertwined to understand the choices made. The aim of training activities is to teach about past experiences, but some search could be done to retrieve the decisions made that will be used in learning activities. Some of the sub-activities defined in the tables described before are interrelated or even duplicated because certain tasks in the producer side are enacted in the consumer side and vice-versa. Figure 1 describes the relationships between the activities defined in the tables and different users can participate either as consumers and producers, depending on their specific roles.

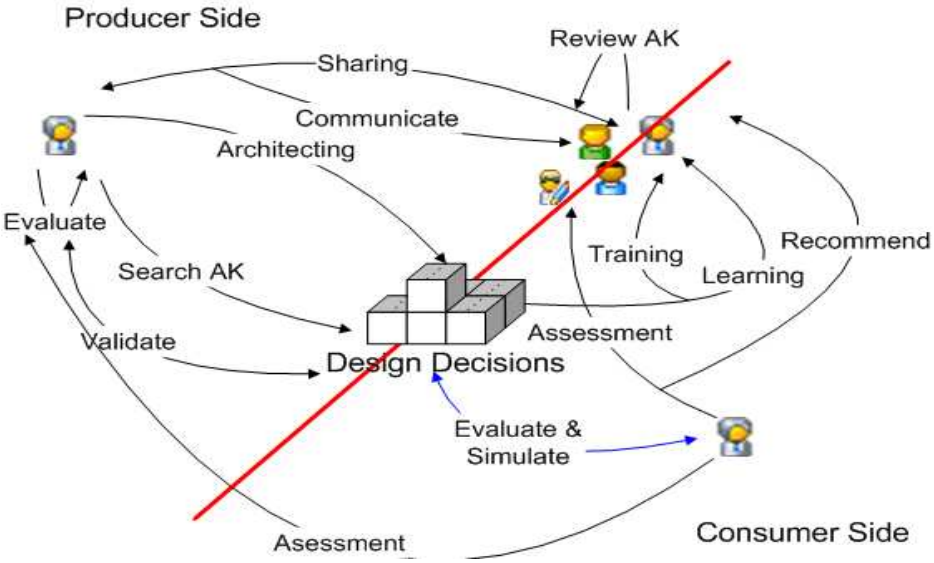


Figure 1. Activities for producing and consuming architectural knowledge

4. Making AK Explicit with Tool Support

Previous efforts [10] analyzed tool support for design decisions in software architecture. Current technology for supporting such AK is still young and immature, but recent proposals are rapidly gaining popularity to introduce design decisions within the architecting process. Some of the tools that have been recently proposed to store and use design decisions are the following.

Archium (http://www.archium.net) is a research prototype [9] for supporting 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* to 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. Archium integrates an architectural description language (ADL) with Java to describe the elements from a component & connector view but making explicit the

architectural design decisions and its rationale [11]. Archium supports the trace from requirements to decisions and is able to check which of these requirements are addressed by one or several decisions. Archium provides visualization facilities for the decisions made using a dependency graph, which can be used to assess about the consequences of the decisions.

PAKME [1] is a web-based architecture knowledge management tool for providing knowledge management (KM) for software architecture development. PAKME has been built on the top of Hipergate, an open source groupware platform which includes collaborative features, project management facilities and online collaboration tools for decentralized teams. At present, PAKME consists of five components: the *user interface* implemented with JSP and HTML pages, the *KM component* which provides the services necessary to store and update AK, the *search component* which defines three different searching mechanisms (i.e.: keywords, logical operators, and navigation) for retrieving artefacts, the *reporting component* which provides services for representing AK and describing the relationships between different architectural artefacts, and the *repository management* which offers the services needed to maintain the data (currently implemented in PostgreSQL). PAKME uses different templates for capturing and representing the knowledge and the rationale associated to architectural design decisions.

The Architecture Design Decision Support System (ADDSS), available at, http://triana.escet.urjc.es/ADDSS) [4] is an open web-based tool developed in PHP, HTML and MySQL, and focuses on recording, managing and documenting architectural design decisions under an iterative development process. ADDSS follows the natural way in which architects usually work, that is, creating the architecture under successive iteration for which one or several decisions are made. The design decisions are stored in plain text in MySQL databases. For each set of decisions, an image of the architecture can be uploaded as a thumbnail image. ADDSS does not directly cooperate with other modelling or requirements tools, but it allows uploading images exported with architecture modelling tools. In ADDSS, decisions are motivated by the requirements already stored in the tool. Also, basic dependencies can be established between a decision and previous ones, as a way to create a network of decisions. The result of the decision-making process can be easily visualized and the user can navigate and browse both the resulting architectures and the decisions made. Design decisions in ADDSS decisions can be based on the selection of well-known patterns already stored and a free text description is used to explain the decision made. Finally, PDF documents containing the design rationale of the architecture can be automatically generated using the *fpdf* library for PHP.

4.1 New Features in ADDSS 2.0

The need to count with adequate tool to support new features for characterizing AK, led to evolve the first version of ADDSS. Therefore, we have recently released ADDSS 2.0 with the following additional features respect to the previous version.

- **Visualization capabilities improved**: In ADDSS 2.0, up to 5 architectures are visualized per row showing the thumbnail images of the architectures with the same width, so users can now browse more easily the architectures across the iterations. Figure 2 shows an example of the iterations list.

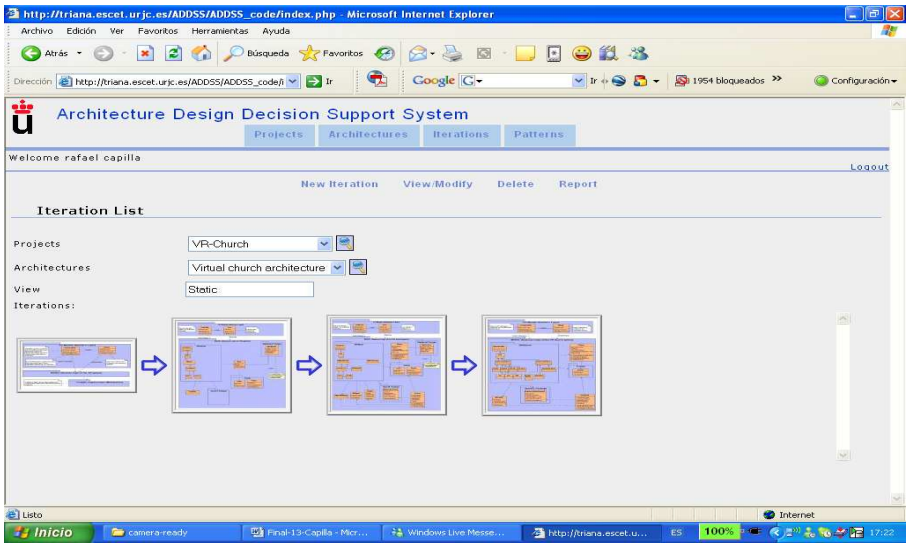
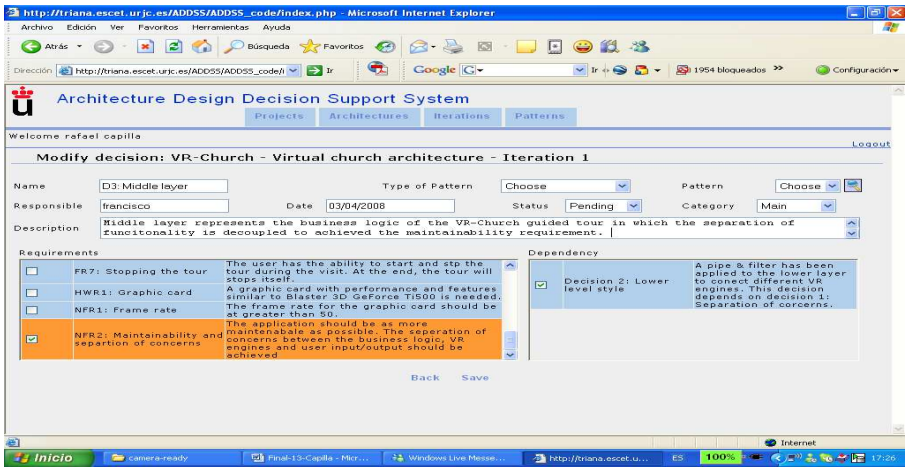


Figure 2. Iterations list shown the architecture products with ADDSS 2.0

- **Status of the decisions**: A status can be assigned to each decision (e.g.: pending, rejected, approved, obsolete), so the architect can know which is the current status of that decision in the project.
- **Date** of each decision can be added.
- **Support for alternatives decisions**: Decisions can be marked as alternative decisions until the final decision is made (one or more decisions could be the best ones).
- **Tagged requirements** as they have been used by a decision. Therefore, the architect knows at every time the amount of requirements that have been addressed during the architecting activity (see Figure 3).



- **Category of the decision**: A category attribute discriminate between main, alternative, and derived decisions. A derived decision has a parent decision.

Figure 3. A design decision with its date, status, the requirements that motivated the decision; and a dependency link to a previous decision

chain of the links between different decisions, so we can easily know which decisions depend from other decisions.

- **User interface improved** (e.g.: menu options, colours).
- **Support for different stakeholder roles**.
- **Pattern classification into different categories**: Pattern search is now more easy and intuitive for the architect.
- **Support for different architectural views**: Now we provide support to define different architectural views and make decisions for each single view.
- **Knowledge search**: In addition to browsing patterns and navigating across the decisions made, a query module extracts relevant information about the decisions made following the links between requirements, decisions, and architectures. For instance, we can extract the requirements and the architectures affected by a particular decision, or we could even know the decisions that affect a particular architecture product.

4.2 Decision-making Process with ADDSS 2.0

According to the activities described in tables 1 to 4, this section describes which of these are implemented in ADDSS 2.0. Table 5 shows in yellow the activities currently supported by ADDSS 2.0. Those activities marked with "+" can be supported by ADDSS and they have been added with respect to the initial classification of section 3 as a refinement of similar tasks. Also, those processes marked inside a dotted box are not directly supported by ADDSS 2.0 (we don't have an explicit attribute to record such information or process implemented to provide some degree of automatic support), but the result of these activities can be stored as part of the description of the decision as a free text description. The remainder activities are not supported by the tool. The tool provides a semi-automatic support to manage the tacit knowledge and make it explicit to users. The explanation of the activities of table 5 supported by ADDSS 2.0 is as follows. During the architecting process, ADDSS 2.0 records the decisions and assigns to them a status as well as other items like the date and the responsible of the decision. The architect can tag a decision as alternative, derived, or main (the selected decision). This reasoning process implies to consider the pros and the cons of any decision, as well as constraints and dependencies between decisions. The reuse of existing AK is limited by this moment to design patterns previously stored. Reusing previous decisions can be done by examining the documentation generated by the tool. The evaluation of the alternatives is externally done but the results are stored in ADDSS in the form as

approved or rejected decisions. Users can navigate through past decisions or even query the database to extract trace information between decisions, requirements and architectural elements.

Table 5. Decision-making activities which are automatic or manually supported by ADDSS 2.0 to record and document relevant architectural knowledge

Decision-making activities supported by ADDSS 2.0		
Activity	Sub-activities level 1	Sub-activities level 2
ARCHITECTING (*): Creates and stores AK		
Make decision	Reasoning (rationale, motivation) Select the best alternative	Constraint and dependency analysis Make assumptions Evaluate AK Analyze implications Validate before storing
Characterize decision	Assign status and other relevant items	
Store and document decisions		
Evaluate AK	Reuse AK Evaluate alternatives	Search, Discovery Navigate through DD (+) Query DD (+) Assessment / Learn
SHARING (*): Make AK available to others		
Review AK	Analyze documents or existing AK stored	Search, Discovery Navigate through DD (+) Query DD (+)
Communicate AK	Subscribe to AK Organize meetings	Pull/ Push (RSS) Discuss / explain
ASSESSING (*): Recommends the selection of a decision		
Evaluate	Evaluate impact of implications Constraint analysis Evaluate impact of quality attributes	Analysis of alternatives Simulation Impact analysis
Review	Check for completeness and correctness of AK	
Validate	Check decisions against requirements and architectural products Check the integrity of the dependencies between decisions	Traceability
Recommend	Communicate to stakeholders the results of the assessment activity	
LEARNING (*): Understand why decisions were made		
Evaluate stored AK	Compare the decisions to products and requirements Detect wrong decisions or inconsistent AK	Follow trace links Search-Reuse AK Navigate through DD (+) Query DD (+)

Training	Teaching about past decisions and experiences	Search-Reuse AK Assessment / Learn

Sharing activities could be partially supported in ADDSS 2.0 by the analysis of existing PDF documentation or stored patterns as well as codified architectures and decisions.

Assessment activities can be supported using the traceability mechanism to check requirements against decisions and validate the decisions made. Also, the results of an evaluation of the alternatives can be stored using the status attribute, but no support is provided to carry out the evaluation process in itself. The basic dependency model supported by ADDSS serves to establish links between requirements and architectures which becomes useful for maintenance and evolution activities.

Finally, learning activities can be only carried out through out the evaluation of the decisions that have been recorded. We can compare the decisions made against the requirements to know how many of these have been addressed, and also trace such requirements to the architectural products developed in the process. The documentation generated by the tool shows the chain of the links between decisions as a way to track manually root causes or even known the implications in the architecture when requirements changes.

Otherwise, inconsistencies or wrong decisions may cause to remove a decision or to mark this as wrong. One key aspect not currently supported happens when we remove a decision. ADDSS does not warn about the consequences of removing a decision, which may cause a broken link in the dependency network. Detecting wrong and inconsistent knowledge is still a challenge to face.

4.3 Impact on Traditional Architecting Activities

Software architecting is considered a formal software engineering approach aimed to create and maintain the architecture of a software system over time. Complex and less complex approaches in combination with other software engineering practices are often used to achieve a balance between the more formal activity of well established methods and the agility required to meet the project schedule. In close relationship to this, the introduction of a complementary and concurrent activity like the creation and use of architectural design decisions with specific tool support changes the traditional way in which software architects do their job. By making explicit the process that records the tacit knowledge residing in the architect's mind, we clearly overload the effort spent by architects in the traditional modeling activity. Recording the design decisions introduces an extra effort in architecting, but a significant reduction should be expected during the system maintenance and evolution, as software architects will be able to replay past decisions as well as to avoid other maintenance tasks like architecture recovery or reverse engineering processes. With ADDSS 2.0 we have tried to balance the processes aimed to store and use architectural knowledge with respect to the more traditional architecting activity. Because ADDSS 2.0 is not integrated with other modeling tools like Rationale Rose, decisions can be stored in parallel at the same time the designers use these modeling tools to depict the architecture. In figure 4 we represent the influence of design decisions in the

potential overhead and reduction effort in architecture development and maintenance phases.

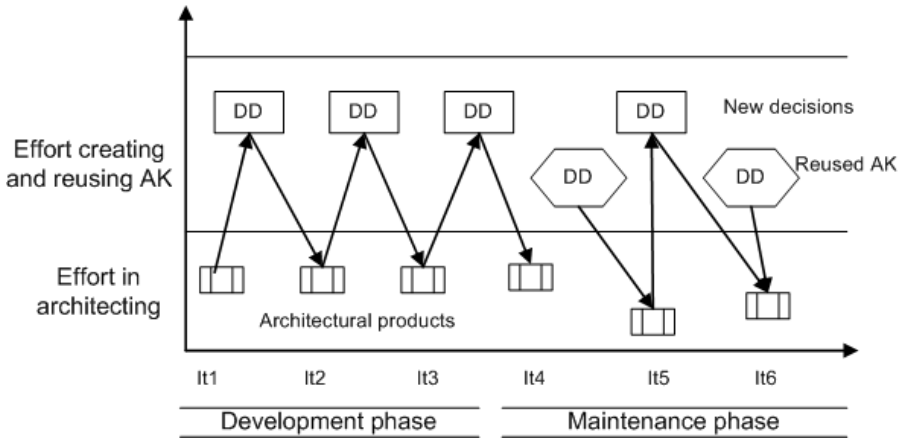


Figure 4. Effort overview extending the traditional architecting activity with explicit decision-making processes for recording and using architectural design decisions

Initially, architects spend a certain effort in creating the architecture during several project iterations (It), and some additional effort has to be made to create the design decisions (DD) including evaluation, assessment, pattern usage, etc. During any maintenance activity, new decisions have to be made while others can be reused (hexagons in figure 4). For instance, the architecture of iteration It6 is the result of a reused decision and a new one. Hence, the effort spent in re-architecting the system is expected to be lower than if decisions were never recorded. Computing this effort is quite important to estimate how much effort can be saved.

5. Conclusions and Future Work

As mentioned in [20], "c*reating and maintaining this rationale is very time-consuming*". At present, we have no empirical data concerning the overhead associated with recording and using architectural design decisions. Because ADDSS 2.0 has just been released, we only have the results from a previous evaluation done with ADDSS 1.0, in which 22 master students participated in the evaluation of a small-medium size project. The students were organized in teams of two persons and they spent around 20 hours to record the decisions of a small virtual reality system which has been modeled using Rational Rose and MagicDraw. Because ADDSS 1.0 has limited features (e.g. no support for decision status or alternative decisions) compared to version 2.0, the main results from the evaluation forms and interviews with the team members can be summarized as follows. Most of the teams perceived ADDSS as easy to learn and use, and they have praised ADDSS for understandability.

Also, depending on experience of the teams, 4 teams spent around 20 hours while 3 teams spent between 7 and 10 hours, and 4 teams took less than 7 hours using the tool. The average time spent by the teams on recording the design decisions was about 10 hours (it does not comprise the traditional modeling activities). Finally, the average scores of the evaluation of ADDSS by the teams ranged between 5 and 10 points in a scale from 0 to 10, except the learning effort that was around 4 points. With respect to the traditional approach, the teams perceived they needed some extra effort to record and maintain the decisions stored in ADDSS 1.0, but we didn't perform cross-comparison creating the same architecture without using ADDSS.

At present, we have performed just one experiment to estimate the overhead associated with recording design decisions. For the next months we expect to have some additional measurable data using ADDSS 2.0 to evaluate the improvements made and estimate the savings when reusing architectural design decisions. Also, we wan to analyze the barriers and the effort needed as we change the traditional way of architecting when recording decisions in parallel with modeling tasks. Because ADDSS tries to bridge the gap between products and requirements, the maintenance phase can benefit from our approach. Moreover, integration with other popular software engineering tools could reduce the effort in capturing decisions.

Finally, the documentation generated extends the traditional architectural documentation and provides valuable information for different stakeholders who want to learn how the architecture was created. Such information crosscuts the information from other architectural views, such as mentioned in the "decision view" [7], which should be seen as a complementary view to the other traditional ones. ADDSS uses plain text in database fields and PDF documents to store and present the design decisions. However, it is planned to export this information to XML documents in order to facilitate the information exchange with other platforms and tools.

References

1. Babar, M. A. and Gorton, I. A Tool for Managing Software Architecture Knowledge. Proceedings of the 2nd Workshop on Sharing and Reusing Architectural Knowledge, ICSE Workshops, (2007).
2. Bass, L., Clements P. and Kazman R. Software Architecture in Practice, Addison-Wesley, 2nd edition, (2003).
3. 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).
4. Capilla, R., Nava, F., Pérez, S. and Dueñas, J.C. A Web-based Tool for Managing Architectural Design Decisions, Proceedings of the 1st Workshop on Sharing and Reusing Architectural Knowledge, ACM Digital Library, Software Engineering Notes 31 (5) (2006).
5. Capilla, R., Nava, F..and Dueñas, J.C. Modeling and Documenting the Evolution of Architectural Design Decisions, Proceedings of the 2nd Workshop on Sharing and Reusing Architectural Knowledge, ICSE Workshops, (2007).
6. 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).
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. Dutoit A., McCall, R., Mistrik, I. and Paech B. (Eds). Rationale Management in Software Engineering, Springer-Verlag (2006).
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. Jansen, A. and Bosch, J. Evaluation of Tool Support for Architectural Evolution, 19th International Conference on Automated Software Engineering (ASE'04), pp. 375-378, (2004).
11. Jansen, A., van der Ven, J., Avgeriou, P. and Hammer, D.K. Tool Support for Architectural Decisions, 6th Working IEEE / IFIP Conference on Software Architecture (WICSA 2007), pp. 4, (2007).
12. Kruchten P. Architectural Blueprints. The "4+1" View Model of Software Architecture, IEEE Software 12 (6), pp.42-50 (1995).
13. Kruchten, P., Lago, P., and van Vliet, H., T. Building up and Reasoning About Architectural Knowledge, QoSA2006, LNCS, pp. 43-58 (2006).
14. Lago, P. and Avgeriou, P. First Workshop on Sharing and Reusing Architectural Knowledge, ACM SIGSOFT Software Engineering Notes, 3(5), 32-36.
15. Perry, D.E. and Wolf, A.L. "Foundations for the Study of Software Architecture", Software Engineering Notes, ACM SIGSOFT, October 1992, pp. 40-52.
16. Roeller, R., Lago, P., van Vliet, H., 2006. Recovering Architectural Assumptions. The Journal of Systems and Software 79, 552-573.
17. Rozanski, N. and Woods. E. Software Systems Architecture: Working with Stakeholders Using viewpoints and Perspectives, Addison-Wesley (2005).
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. van der Ven J.S., Jansen, A.G., Nijhuis, J.A.G., and Bosch, J. Design Decisions: The Bridge between the Rationale and Architecture. In Rationale Management in Software Engineering, pp. 329-346, Springer-Verlag (2006).
21. Wang, A., Sherdil, K. and Madhavji, N.H. ACCA: An Architecture-centric Concern Analysis Method, 5th IEEE/IFIP Working Conference on Software Architecture, (2005).