Effort Estimation in Capturing Architectural Knowledge

Rafael Capilla[1], Francisco Nava[1], Carlos Carrillo[2]
[1] *Depto. de Ciencias de la Computación, Universidad Rey Juan Carlos, Madrid, Spain*
[2]*Depto. de Ingeniería y Arquitecturas Telemáticas, Universidad Politécnica de Madrid, Spain*
{rafael.capilla, francisco.nava}@urjc.es, ccarrillo@diatel.upm.es

Abstract

Capturing and using design rationale is becoming a hot topic for software architects, as architectural design decisions are now considered first class entities that should be recorded and documented explicitly. Capturing such architecture knowledge has been underestimated for several years as architects have been only focused on documenting their architectures and neglecting the rationale that led to them. The importance of recording design rationale becomes enormous for maintenance and evolution activities, as design decisions can be replayed in order to avoid highly cost architecture recovery processes. Hence, in this work we describe how architecture design decisions can be captured and documented with specific tool support. We also provide effort estimation in capturing such knowledge and we compare this with architecture modeling efforts in order to analyze the viability of knowledge capturing strategies.

1. Introduction

For many years, software architectures have been proven useful for representing the main parts of a software system [4] by means of architectural descriptions or diagrams. From the nineties, different types of descriptions (mostly based on UML diagrams) are used to provide different perspectives or architecture views for different stakeholders [9, 16, 20]. Recently, the software architecture research community as well as software architects from the industry are facing the problem to consider architecture design decisions as first class entities that should be documented explicitly [6]. The importance of design rationale in software architecture was early stated in the nineties by Perry and Wolf [18], which consider the rationale as a relevant piece for understanding the design. More recently, Kruchten et al. [17] have modernized this idea as they state that Architectural Knowledge (AK) = Design Decisions + Design. Hence, the importance of design rationale that has been neglected in the past becomes now relevant for most modern architecting processes.

Therefore, recording, using, managing, and documenting architectural design decisions are new complementary activities (e.g.: capturing knowledge, sharing) that should be carried out in parallel to typical architecture modeling tasks. These new challenges need to deal with many obstacles in order to overcome those barriers that try to impede the transfer of implicit mental models from the architect's expertise to explicit and documented knowledge. This architectural knowledge (AK) should be codified in a suitable form that can be reused afterwards when needed.

The goal to document explicitly undocumented knowledge has an overhead that should be taken into account if we want to estimate the potential savings in typical maintenance and evolution activities. In this context, having tool support to provide some degree of automation for the activities aimed to capturing this knowledge becomes a strong need as a mean to facilitate the gradual introduction of documented design rationale in typical architecting processes.

The remainder of this work is structured as follows. Section 2 describes the motivation of our work as well as the ADDSS approach, a web-based tool for capturing and documenting architectural design decisions. In section 3 we describe the experiences carried out with ADDSS and the results we obtained measuring the effort in capturing architectural design decisions. Section 4 discusses related work and section 5 outlines the conclusions and future work.

2. Capturing AK with Tool Support

In this section we first introduce the motivation of our approach and after we describe the ADDSS tool for capturing and documenting architecture design decisions.

2.1. Motivation

Software architects have widely used architecture modeling tools for producing and documenting the models of their system's architecture. At present, there is lack in the documentation generated by typical architecting processes as they never record the design decisions that led to a particular architecture. This problem is slightly mentioned in [9], which states the importance for recording design rationale but not the processes required to produce such knowledge. In the past, typical architecting tools don't include design rationale as a first class entity that has to be documented and only one of the five tools discussed in [13] (i.e.: Compendium) provides limited support for capturing first class architectural design decisions (maybe some of these tools can evolve to more advanced versions in which their capabilities are extended to support new concepts and features).

In [24] the authors mention several reasons for using rationale in software engineering like supporting knowledge transfer or improving quality. Design decisions can bridge the gap between rationale and architecture whilst the rationale enriches architecture with the underlying reasons that led to them. Thus, design rationale has a strong impact in the architecting construction process, and the design choices can play a key role if these can be captured and used.

Another barrier that has to be tackled is the mentality by which users (e.g.: architects, business managers, developers) consider enough important the value of design rationale and how they use and document the knowledge related to design decisions. The survey described in [21] mentions the lack of empirical evidence to probe the designer's perception for documenting such valuable asset. However, barriers found for using and documenting design decisions and its rationale motivate the use of processes and tool support to facilitate the gradual introduction of new roles in architecting like producers and consumers of architectural knowledge. These new practices need of specific tool support for using and documenting the design rationale and avoid negative designer attitudes as they don't care to spend effort in capturing such AK. A complementary experience carried out in [12] tries to probe the value of design rationale in software architecture. This approach mentions that users enacting different use cases need different type of information describing the design rationale as this information can be tailored to different needs.

As a result, we can state that there are many cultural and technical barriers that may hamper the use and documentation of design rationale and users need to be convinced about the expected benefits in doing new complementary activities pertaining to the architecture construction process.

One way to motivate the capture and use of AK is to probe the expected benefits in further maintenance and evolution activities. Hence, the overhead required in the development phase can be compensated during maintenance. Is a well-known fact in software engineering that maintenance is a time consuming activity that requires a lot of effort and time in order to support the evolution of systems over time. For instance, typical maintenance costs may range between 50-75% of the total development cost [5]. The cost for developing and maintaining the architecture should be also considered as important as well. Often, maintenance is only performed at the code level and when the architecture becomes obsolete or inexistent, highly cost reverse engineering processes have to be carried out to recover the design from the changes made in the code, as the design decisions that led to such changes were never recorded. We have motivated the importance for recording architectural design decisions in relationship to typical architecting activities. Also, estimating this new effort is important to know how recording such design decisions can pay off. Hence, we need to probe the value of capturing the design decisions in the overall design process, as effort estimation will serve us to know the savings along the life of a software system.

2.2. The ADDSS Approach

The Architecture Design Decision Support System (ADDSS – http://triana.escet.urjc.es/ADDSS) is a web-based tool for capturing, managing and documenting architecture design decisions [7]. ADDSS' conceptual model [8] relies on the ideas described in the "decision view" [11], which consider this as a new cross-cutting architectural view with respect to the traditional ones [16] used for documenting the architecture. The decisions view fosters capturing and documenting the design decisions and the rationale that happen during any architecting process.

The main capabilities of ADDSS are summarized in the following bullet points.

- *Capturing design rationale*, which provides templates of attributes specific to capture and record the design decisions and their rationale. Mandatory and optional list of attributes provide flexible approach for characterizing and documenting the design decisions for different user profiles.
- *Stored design patterns and architecture styles* which can be retrieved as design solutions during the reasoning activity.
- *Relationship between requirements, architectures and decisions*, are useful links that can be established during the architecting activity to bridge the gap between requirements and architectures. Also, links between decisions can be defined to track the traces between the decisions and used to estimate change impact analysis or to track the root causes of changes.
- *Iterative construction process* comprises the main architecting activities which relate typical architecture modelling tasks with the characterization of the design decisions. In this phase, we construct and visualize the different architecture products with their decisions. Architectures can be uploaded into ADDSS knowledge base to show the evolution of the design. ADDSS users can navigate and brose the architectures and the decisions made.
- *Architecture view support* which provides the different perspectives of the same architecture. Users can visualize each different architecture view and display the UML diagrams for each single view.
- *Documenting decisions*, results key to extend the traditional architecture documentation view following the goals stated in the "decision view" [11]. An automatic report facility provides online PDF documents containing a detailed description of the decisions made for each architecture, as well as the requirements and architectures for each set of decisions. This documentation clearly shows the relationships between decisions and requirements and architectures.

Not all the activities dealing with architectural design decisions are supported by ADDSS. The previous list provides a set of core features that seem to be enough for capturing design decisions, as it is the mail goal of this work. Otherwise, are capabilities are desired for future extensions of the tool, like for instance knowledge sharing. Also, some improvements to previous approaches have been made, such as the feature of ADDSS that describes the evolution of the architectures with their design rationale in an iterative way, similar as architects built their architectures.

3. Effort in Capturing Design Rationale

In order to face some of the challenges described in section 2.1, we used ADDSS to determine the effort in capturing architecture design decisions compared to typical architecture modeling tasks. Hence, we can estimate the overhead required to store such knowledge and determine the viability of capturing AK as well as the potential benefits for further maintenance activities.

We first summarize previous experiences using ADDSS and second, we outline how we carried out a case study to estimate the effort required to capture architectural design decisions in different phases of the software life-cycle.

3.1. Initial experiences using ADDSS

We released ADDSS 1.0 in 2006 and version 2.0 in 2007, and we evaluated the tool by means of several experiences as we summarize below.

3.1.1. ADDSS 1.0 with URJC students. We used ADDSS 1.0 with twenty-two master students from a master course of the Rey Juan Carlos University(URJC), Madrid (Spain), participated in the evaluation. The students were organized in eleven teams of two persons and they had to store the decisions and the architectures for a subset of requirements belonging to a virtual reality system (VR-Church). The students were familiarized with software architecture concepts and architectural design decisions before using ADDSS. As a result, we interviewed the students and all of them had to fill a questionnaire to evaluate the capabilities of the tool. The average effort spent by all the teams in capturing the decisions was 10 hours, with some significant differences among the teams, maybe because of the different experience of the members, as many of them work for companies. Other results highlighted the high usability of the tool, the easy of use, and the low learning effort. The students employed between four and six iterations to deliver the final architecture of the VR-Church system, and several intermediate architectural products were stored with the decisions made. Figure 1 shows the results of the effort in hours spent by all the teams.

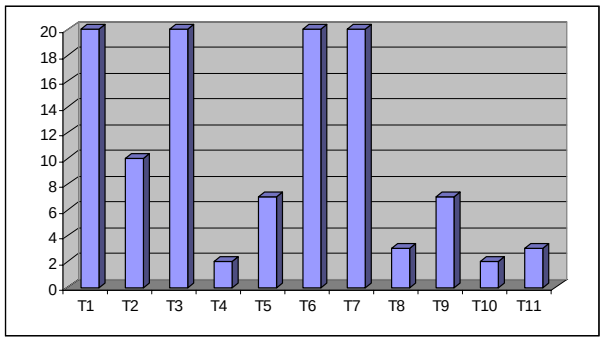

Figure 1. Effort spent by the teams using ADDSS

3.1.2. Architecting a virtual reality system. In our second experience we used ADDSS 2.0 (released in 2007) with improved capabilities with respect to version 1.0. Some of these new capabilities include: *better architecture visualization, support for architecture views, support for alternative decisions, and attributes to describe the status and category of the decisions stored*. To test the new version we used the same VR-Church system but taking into account the complete list of requirements instead of the subset used in our first case study. In the experience two of the co-authors acted as software architects while two other persons acted as domain experts.

Interviews with the two domain experts and a domain analysis activity were carried out before using ADDSS. We spent around one month in the design and modeling activities of the VR-Church system, and four intermediate architecture products were built (with Magicdraw 10.5) before the final architecture was released. Hence, the construction the VR-Church architecture took five iterations, and we stored these using ADDSS, such as Figure 2 shows.

During the decision making process, a number of 32 coarse-grained design decisions, including 15 alternative decisions were considered. From the whole number of decisions, 24 of them were approved as valid. The status and category attributes were proven useful during the evolution of the architecture. Fine-grained decisions like variation points were not stored in order to alleviate the effort in capturing the decisions. We also defined and stored 24 dependencies between the decisions. We didn't measure the effort in recording such knowledge but we proved the viability of the tool for capturing and documenting the design decisions along with their architectures. Figure 3 shows a screenshot used by the architect to capture the information of a design decision, with explicit links to the requirements that motivate such decision (left box of Figure 3), and dependencies to previous decisions, shown in the right box of Figure 3.

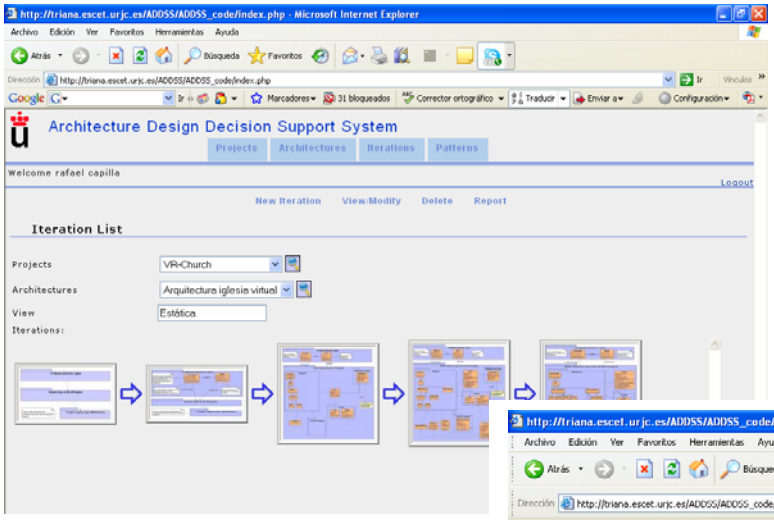

Figure 2. ADDSS iterative architecting process

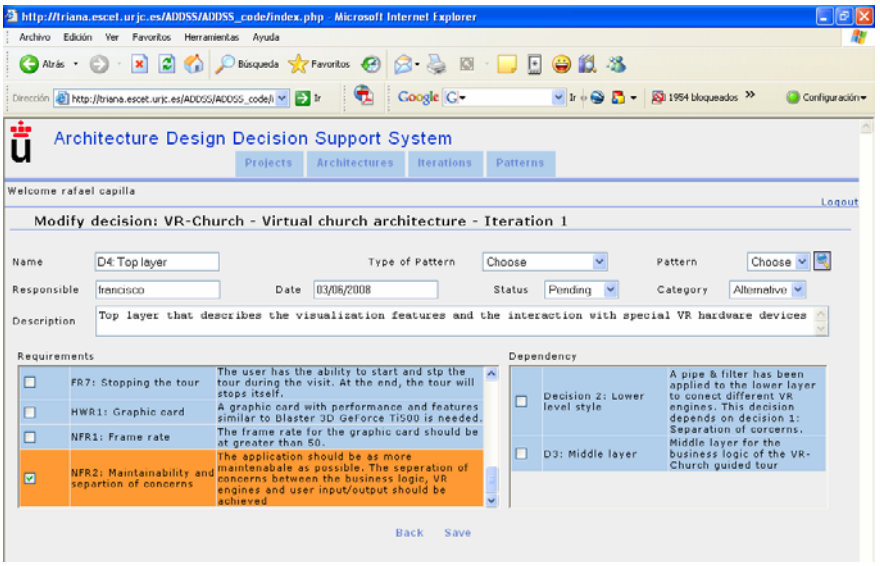

Figure 3. Characterization of a design decision with explicit links to requirements and architectures.

Once the architect has characterized a design decision, he/she can browse the list of decisions made (Figure 4) and upload the architecture, which is the result of the set of decisions made. After the decisions are stored for the first time, the architect and other relevant stakeholders cans deliberate about the alternative decisions and after making the right choices, change the status and the category in ADDSS of the design alternatives considered.

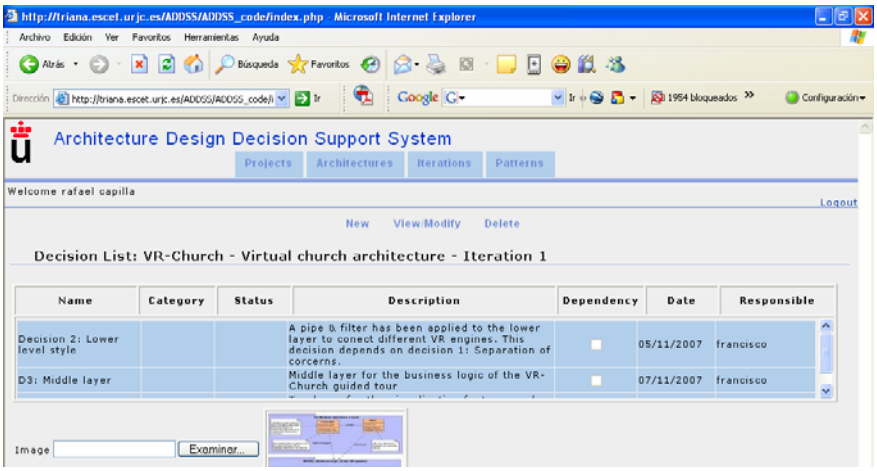

Figure 4. List of decisions that motivated a particular architecture.

3.1.3. ADDSS at the Fraunhofer IESE. During June 2007 we tested ADDSS in collaboration with the Fraunhofer Institute for Experimental Software Engineering (IESE) in Kaiserslautern, Germany. We combined ADDSS with a reverse engineering tool developed at Fraunhofer IESE called SAVE, (Software Architecture Visualization and Evaluation, [15]) to document the design decisions made over a different tool. One of the co-authors and one software engineer from the Fraunhofer participated in the experiment.

The SAVE tool was applied to analyze the evolution of an existing tool (i.e.: DecisionModeler) to recover the architectures of the changes made in the code of the DecisionModeler. Because SAVE is unable to describe the rationale of the changes made in the DecisionModeler, we used ADDSS to store and document such decisions as well as the main architectures recovered with SAVE. We employed 4 iterations to record the main architectures recovered with SAVE. We didn't store the intermediate architectures corresponding to minor changes (i.e.: small refactorings) of the functionality of the DecisionModeler tool. ADDSS was useful and complementary with the outcome of the SAVE reverse engineering tool; but to scale-up ADDSS to industrial applications, the Fraunhofer recommended to extend the multi-user management features and a way for resolving overlapping or incompatible decisions. As a result from this combined experiment we captured and documented the design decisions of the architectures recovered using SAVE. We experienced the benefits for replaying past design decisions and we perceived these useful in future refactorings of the DecisionModeler tool.

Also, recorded decisions can be helpful to align them with changes made at the design and code levels. We also observed that micro-architectural decisions would take more documenting effort than coarse-grained decisions, but for our purpose we considered enough to store the main decisions and their architectures belonging to the major changes performed over the DecisionModeler.

Generally speaking, software architects usually do their reasoning activities implicitly in their minds, but partially automating such activities (e.g.: the automatic documentation facility of ADDSS) and recording this valuable knowledge, results hard because we are trying to change the way in which architects do their job, as we want to describe explicitly the decisions that are implicit in their minds. These new tasks should run in parallel with modeling tasks and they represent an overhead that has to be estimated to know the potential benefits for further maintenance operations. In order to demonstrate how much effort would be required to record this architectural knowledge, we carried out a recent experiment, such as we describe in next section.

3.2. A case-study to estimate the effort in capturing AK with ADDSS 2.0

During January 2008 we carried out a new experiment to estimate the effort in capturing design decisions with ADDSS 2.0 along different phases of the software life-cycle.

3.2.1. Design and objectives of the case study.

In the experience, we didn't follow the typical guidelines for controlled experiments. Instead, we preferred to give the students the chance to capture the design decisions out of the laboratory hours. The motivation for this relies on the future collaborative capabilities of ADDSS that can be used by distance and distributed teams to store and discuss the decisions remotely at different times. Hence, our experiment tries also to simulate this kind of situations on behalf of the nature of ADDSS as a web-based tool. Because we are interested in maintenance and evolution, we designed the experiment in three different phases: *development*, *maintenance, and evolution*, and we measured the effort spent in both reasoning and modeling tasks for the three phases. By doing this we could estimate the overhead required in capturing the decisions with tool support with respect to typical architecture modeling activities. We used ADDSS 2.0 with 17 master students from a regular computer science course from the Rey Juan Carlos University (URJC) of Madrid (Spain). At least around 50% of the subjects work for software companies and they can be considered as senior software engineers. All of them possess concepts on software architecture and they have been trained on the notion and use of architectural design decisions with ADDSS. In our case study we used the same VR-Church system as the target system and all the subjects had to capture the decisions for the three phased mentioned before.

For the *development phase* we gave them 16 requirements from the VR-Church system and 5 tables containing the design decisions already made before by other software architects, including the alternatives and the choices made as well as the rationale that motivated such decisions. A number of 32 design decisions including 15 alternatives were given to them. The students had to replay the development phase storing in ADDSS the information concerning the design decisions and measuring the time employed in this task. Each of the tables containing a subset of the decisions belongs to a single iteration in the architecting process, as ADDSS supports an iterative process that clearly shows the evolution of the architecture over time.

In addition, they had to specify the architectures for the decisions stored for each of the iterations enacted. They used the UML Magicdraw tool in their modeling activities and they measured the time spent in such modeling tasks. Users only had to model the static view of the architecture in the form as a UML package and class diagram as other architecture views were not considered. When capturing the decisions, the users had to simulated the reasoning mental activity and store the alternative decisions including a status (e.g.: pending) and a category (e.g.: alternative). After that, the simulation of the mental process for selecting the design choices implied a change in the status to approve or reject the decisions and to select its category (e.g.: main, alternative, and derived). Hence, the time measured in this activity tried to simulate how the architect would reason when several alternatives are available and taking into account the rationale behind such decision.

Once the first 5 architectures were stored as result of the development phase, we started the *maintenance* and we gave the subjects a new set of requirements (7 FR, 1 HWR, and 1 NFR). In this phase the subjects had to make new design decisions (not already made before) and model the architectures. In addition, we gave them the freedom to use the architecture from the last iteration of the development phase and modify the existing decisions or to create new iterations to store the new decisions and the resulting architectures.

Finally, we carried out an *evolution phase* in which the architecture had to evolve to a better stage. In this phase users were told to refine the previous decisions and architectures by adding fine-grained decisions belonging to the specification of new class attributes, methods, and variation points into the UML classes. Hence, subjects had to revisit many of the decisions made in the development and maintenance phases and modify some of these to add new ones, in particular with a lower granularity level. Another possibility they had was to create new iterations to store the new decisions, as well as reflecting such changes in the designs. We also told the students to gather and report the effort spent in their reasoning and modeling activities and to provide personal conclusions from the experiment. ADDSS provides an automatic reporting system that produces PDF documents of the information stored. The students performed the three phases individually, but they could interact between them to exchange ideas or to discuss about the suitability of a potential design choice, even with the instructor. Hence, the time employed in the peer-to-peer discussions was added to effort spent in their reasoning activities. In next subsection we outline and discuss the results we obtained.

3.2.2. Evaluation. After analyzing the results, we did a preliminary evaluation and we discarded the results of 3 subjects because they failed in doing correctly experiment. Hence, we accepted 14 results out of 17 as valid. Because our main goal was to estimate the overall overhead of the effort spent in capturing and maintaining the design decisions for the three main phases, we didn't analyze individually the results for each subject to know how many decisions were captured for each iteration. The results we obtained for the three phases mentioned above are discussed below.

Development (architecting): The overall effort we measured from the subjects in capturing the design decisions and in the modeling tasks during the initial architecting phase is shown in Figure 4. The 14 subjects spent a total of 1759 hours in the reasoning and capturing of the design decisions for the first five iterations of the architecture development phase and 2006 hours in modeling the five architecture products. Hence, the subjects spent a total of 247 hours in their modeling tasks more than in the decision-making activities. Capturing the decisions consumed less effort than typical modeling activities as the decisions were already made by others and given to the students. The overhead of 1759 hours spent in the reasoning activities represents a 47 % with respect to the traditional architecting phase. In addition, the users defined an average of 23.5 dependencies between design decisions. One of the subjects established a high number of dependencies (i.e. 95), two subjects didn't establish dependencies, and three of them defined less than 10 dependencies. The other subjects were on the average. Therefore, we could deduct that some of the subjects misunderstood the notion of dependency between design decisions and confuse this with dependencies between requirements or even between parts of the architecture.

Maintenance: During maintenance, the effort spent in capturing the decisions decreased as we expected, but compared to the architecting phase, the students spent more time in their reasoning activities than in the modeling ones (i.e.: 918 hours for making and capturing decisions and 715 hours in modeling tasks, as Figure 4 shows). This is because in the development phase the decisions were already given to the subjects, while in the maintenance phase the users had to think from scratch in new decisions based on new requirements. Users spent 203 hours more in their reasoning activities than in the modeling tasks. In addition, the subjects defined an average of 13.5 dependencies more than in the development phase, as a result of the complexity introduced by the new requirements which motivated new related decisions and new relationships in the architecture. Moreover, the number of iterations above 5 increased in most cases, as 7 subjects added 2 new iterations, 2 of them 3 iterations, and 4 of them only 1 iteration. One of the subjects didn't add any new iteration and he/she incorporated the new decisions into the existing architectures. The subjects needed also an average of 1.8 iterations to store the new decisions. Hence, with only two iterations the subjects stored in most cases the new set of decisions, and only two new architectures had to be modeled. In this phase the percentage of the effort spent in the decision-making activity (reasoning + capturing) was around 56 %. The effort obtained in this phase is not in contradiction with the total percentage of maintenance effort of a typical software project as we only carried out one single maintenance task during a brief period. Typical maintenance effort percentage is usually estimated over months or years.

Evolution: Finally, we carried out an evolution activity to produce a more detailed architecture with respect to the previous one. Hence, the architecture evolved from its current state to a better one. The evolution phase consisted in detailing the attributes, methods, and variation points in the classes of the UML design. No new concrete requirements were given to the subjects, but only the goal to improve the existing designs. Hence, the subjects had the freedom to revisit previous decisions and add concrete requirements to motivate the inclusion of specific attributes, methods, and variation points (e.g.: by means of stereotypes and tagged values) in the design. As a result, users spent a number of 453 hours in creating and capturing the new decisions and 530 hours in the modeling tasks (see Figure 4). The difference between both efforts is 77 more hours spent in modeling activities. We can explain this difference because most of the new decisions were a refinement of previous ones taken in the development and maintenance phases. Hence, users didn't have to spend much more time in reasoning about new decisions but only modify existing ones or add decisions based on previous knowledge (i.e.: knowledge reused). Users spent around a 46 % of their effort in creating the new decisions for this phase. In this case, adding fine grained decisions was remarkably more easy that capturing the main key decisions for the first stages of the architecture. Because users had to refine the previous decisions, only 17 new dependencies were added, which represents only an increment of 1.2 % with respect to the maintenance phase. In addition, all the subjects excepting two of them

added a single iteration to produce the new version of the architecture with the refactorings made. Hence, we believe it was easier to the subjects to add new decisions in a new iteration than to refine existing ones, which are often split across several iterations and phases. Hence, user had only to improve a single architecture from the last iteration rather than several ones across different iterations. The evolution phase was slightly different than the two previous phases because in the development and maintenance phases, the granularity of the decisions was coarse-grained (e.g.: based on patterns and styles) while in the evolution phase all the decisions were fine-grained. Low level granularity decisions usually introduce more complexity in the network of decisions, but the users didn't perceive this as they only added a few set of new dependencies. The summary of the effort spent in the decision-making and modelling activities is summarized in figure 4.

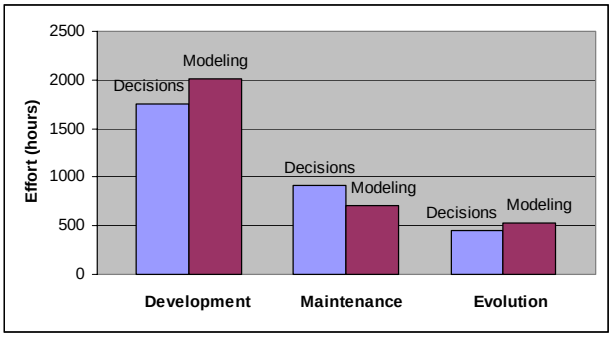

Figure 4. Summary of the effort spent in decision-making and modelling tasks for the three phases

We can observe from Figure 4 that users needed to spend more effort in the development phase for creating the first decisions and modelling the architecture products, even if the decisions have been made before. Hence, this effort is expected to be saved or reduce in subsequent maintenance phases (including evolution) as the time needed to maintain previous decisions decrease significantly.

3.3. Threats to Validity

In this section we discuss some of the issues that might have affected the results of our experiment and may limit the generalizations of the results.

The first issue refers to carrying out a controlled experiment in the sense that we gave the students the freedom to perform the three phases at their houses. Hence, we didn't control the time spent by the subjects in a controlled environment and we had to trust in the results. To partially solve this threat we gave many advices about how to enact the case study and tell them to be fair and honest in measuring the effort.

The second issue involves the decisions of the development phase as these were made before and given to the subjects. The main reason for this was because users were not familiarized with virtual reality system development and we wanted to facilitate them the first architecting stages. This issue is easy to solve by giving the subjects only the requirements that will motivated the decisions in the development phase.

The third issue might affect the generalization and extrapolation of the results as we could replay the same experiment in a different context. Our work cannot be generalized to industrial settings since we conducted it within an academic scope. Performing the same tasks in an industrial setting would be valuable for us in the way as we could deal with other type of systems and address the concerns of software architects in a real environment with different characteristics.

Regarding Figure 4,one could say the effort of doing a project with design rationale would be twice as without storing the decisions, but two important factors have not been measured in this experience. The first refers to the period used for analyzing the results. Evidently, a long evaluation period is needed to estimate better the maintenance and evolution efforts of the design decisions and the savings that could be achieved in the long term with respect to the development tasks. The second refers to the cost, not included in the experiment, of potential reverse engineering activities carried out to understand a particular architecture in the absence of design rationale. Figure 4 provides only effort in time to estimate the burden of capturing design rationale and compare these in three different phases, as a proof that maintenance effort can reduced. Otherwise, we didn't use cost-benefit analysis to estimate the ROI in the experiment.

4. Related Work

Initial attempts for supporting Knowledge Based Software Engineering (KBSE) systems were popular in the eighties. For instance, gIBIS [10] is an application hypertext tool designed to facilitate the capture of early design deliberations with some collaborative features. Most of these attempts were focused on general knowledge and only a few tools have been design with software engineering in mind.

To date, architectural knowledge has been often neglect in software architecture development, but an increasing number of ongoing efforts for capturing and

documenting design decisions and its rationale, some of them with tool support, have produced some promising results since 2004. Basic research has been done for representing the information of architectural design decisions and how this should be captured. Some authors have focused on describing template lists of attributes for capturing such knowledge [8] [17] [23] while others use ontologies to organize the design decisions in addition to the attributes that are used to describe such AK [1] [17]. Others have focused on describing underlying rationale in the form of assumptions [19] that are produced during the reasoning activity in architecting.

Specific tool support is still in not yet mature and at present, some research prototypes in addition to ADDSS have been developed. Most of them are still ongoing projects belonging to different universities. For instance, PAKME [2] [3] is a web-based architecture knowledge management system that captures architectural design decisions and design rationale though specific templates. PAKME is built on the top of an open source groupware platform (Hipergate) and provides collaborative features. Archium [14] is a Java tool that integrates requirements, decisions, architectures, and implementation models, and provides traceability among a wide range of concepts. Archium uses an architecture description language (ADL) to describe the architectures from a component & connector view, and stores and visualizes design decisions and its rationale. AREL [22] is a tool for capturing architecture decisions, design rationale and design options by means of a UML meta-model. In AREL, UML entities are linked to show the relationships between design decisions, design concerns and design options. Evolution history of the decisions is captured with eAREL, which is an extension of AREL.

With respect to empirical studies, the work described by Falessi et al. [12] has evaluated the importance of documenting design rationale based on the expected benefits, but not the effort in capturing such knowledge.

5. Conclusions and Future Work

This work proves the viability of using a codification strategy for capturing architecture design decisions in parallel to typical architecture modeling activities, as opposite to other strategies based on personalization. ADDSS partially automates the capturing ad documenting of architectural design decisions and assists the architect in such engineering activity. The results from the experiment carried out refine previous ones and provide more insight as we have estimated the effort in three different phases of the software life-cycle. In addition, comparing the effort spent in reasoning and capturing design rationale with respect to typical modeling activities let us know to estimate the overhead required for the three different phases, and which of these may require more effort, often depending on the number and type of decisions to store (i.e.: coarse-grained and fine-grained decisions).

The initial architecting phase was a more time-consuming activity when decisions were recorded for the first time. This required overhead can be saved in further maintenance and evolution activities as shown in Figure 4, but we perceived the experienced of the subjects using the tool plays also an important role as training activities can be carried out to accelerate the capturing of knowledge. Also, because maintenance and evolution activities are carried out along many years, a longer study is needed to obtain more precise results, as well as to estimate the ROI taking into account the savings of avoiding other processes like reverse engineering. In addition, it would be useful to estimate the number of decisions reused or which decisions become obsolete.

From the reports the subjects had to fill evaluating the usefulness of the tool, most of them perceived it useful and easy to use. One additional conclusion from the experience out with the Fraunhofer IESE was the utility to count with recorded design rationale, as the architect didn't need to remember past decisions which can be easily replayed and remembered for the refactorings.

For future work we would like to carry out longer experiments to estimate more accurately the savings of and the ROI. Also, we would like to know the impact in the agility of the architecting process when the number of attributes for characterizing a decision increases and how this relates to the user satisfaction when capturing such knowledge. This might affect the effort spent in capturing the design decisions as users could be interested in capturing different types of information. As ADDSS is not integrated with external modeling tools, this may hamper the use of tools like for supporting design rationale. A future integration with other software engineering tools may facilitate the introduction of these new activities in the traditional architecting processes.

6. Acknowledgements

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. We also thank people from the Fraunhofer, Jens Knodel, Dirk Muthig, and Thomas Forster, for

supporting our work and providing useful conclusions for future improvements.

7. References

[1] Akerman, A. and Tyree, J. Using Ontology to Support Development of Software Architectures. IBM Systems Journal, 45 (4), (2006), 813-825.

[2] Ali- 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, IEEE DL (2007).

[3] Ali-Babar, M., Gorton, I. and Kitchenham, B. A Framework for Supporting Architecture Knowledge and Rationale Management. in Dutoit, A.H., McCall, R., Mistrik, I. and Paech, B. eds. Rationale Management in Software Engineering, Springer, (2006), 237-254.

[4] Bass, L., Clements P. and Kazman R. Software Architecture in Practice, Addison-Wesley, 2nd edition, (2003).

[5] Boehm, B. W. Software Engineering Economics, IEEE Transactions on Software Engineering, 10 (1), 4-21 (1984).

[6] 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).

[7] 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).

[8] Capilla, R., Nava, 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, IEEE DL (2007).

[9] 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).

[10] Conklin, J. and Begeman, M., gIBIS: a hypertext tool for exploratory policy discussion. in Proceedings of the 1988 ACM conference on Computer-supported cooperative work, (1988), 140-152.

[11] 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).

[12] Falessi, D, Cantone, G., Kruchten, P. Do Architecture Design Methods Meet Architect's Needs, 7th Working IEEE / IFIP Conference on Software Architecture (WICSA 2007), pp. 5, (2008).

[13] 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).

[14] 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).

[15] Knodel, J., Lindvall, M., D. Muthig, M. Naab. "Static Evaluation of Software Architectures", 10th European Conference on Software Maintenance and Reengineering, Bari, Italy, (2006), 279-294.

[16 Kruchten, P., Architectural Blueprints. The "4+1" View Model of Software Architecture, IEEE Software 12 (6), pp.42-50 (1995).

[17] Kruchten, P., Lago, P., and van Vliet, H., T. Building up and Reasoning About Architectural Knowledge, QoSA2006, Springer-Verlag LNCS 4214, pp. 43-58 (2006).

[18] Perry, D.E. and Wolf, A.L. "Foundations for the Study of Software Architecture", Software Engineering Notes, ACM SIGSOFT, October 1992, pp. 40-52.

[19] Roeller, R., Lago, P., van Vliet, H., 2006. Recovering Architectural Assumptions. The Journal of Systems and Software 79, 552-573.

[20] Rozanski, N. and Woods. E. Software Systems Architecture: Working with Stakeholders Using viewpoints and Perspectives, Addison-Wesley (2005).

[21] 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).

[22] Tang, A., Jin, Y. and Han, J. A Rationale-based Model for Design Traceability and Reasoning, Journal of Systems and Software 80, pp. 908-934, Elsevier (2007).

[23] Tyree, J. and Akerman, A. Architecture Decisions: Demystifying Architecture. IEEE Software, vol. 22, no 2, pp. 19-27, (2005).

[24] 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).