The role of a "decision view" in software architecture practice

Philippe Kruchten[1], Rafael Capilla[2], Juan C. Dueñas[3]

[1]**Department of Electrical & Computer Engineering, University of British Columbia, Vancouver, Canada**
[2]**Departamento de Ciencias de la Computación, Universidad Rey Juan Carlos, Madrid, Spain**
[3]**Departamento de Ingeniería de Sistemas Telemáticos, Universidad Politécnica de Madrid, Spain**
pbk@ece.ubc.ca, rafael.capilla@urjc.es, jcduenas@dit.upm.es

Abstract

A "decision view" provides a useful addition and complement to more traditional sets of architectural views and viewpoints; it gives an explanatory perspective that illuminates the reasoning process itself and not solely its results. This decision view documents aspects of the architecture that are hard to reverse-engineer from the software itself and that are often left tacit. The decision view and the decisions that it captures embody high-level architectural knowledge that can be transferred to other practitioners, merged when systems are merged, and offer useful support for the maintenance of large and long-lived software-intensive systems. This article leads the reader through a succession of epiphanies: from design to architecture, then architecture representation to architecture design methods, and finally to architectural design decisions.

Keywords*: Software architecture, architectural design decision, architectural knowledge, architecture views, decision view.*

1. Introduction

Current software development has to deal with many challenges, such as the increasing complexity of systems, the growing request for quality levels, the burden of maintenance operations, distributed production and high staff turnover. Software companies that strive to reduce the cost of maintenance demand increasingly more and more flexible designs that would be easier to maintain. It is now well recognized that software architecture constitutes the cornerstone of software design, key for facing these challenges. Several years after the software

crisis, the practice of software architecture emerged and it is now a mature, although still growing discipline, capable of addressing the increasing complexity of new software systems. The term ”software architecture” was first coined in the NATO conference on software engineering techniques in 1969, but it was not until the late eighties that software architectures were used in the sense of system architecture [1].

Still today, modern software architecture practices rely on the principles that Perry and Wolf enounced in their pretty and simple formula Architecture = {Elements, Form, Rationale} [2]. The elements are the main constituents of any architectural description in terms of components and connectors, while the nonfunctional properties guide the final shape of the architecture. Different shapes with the same or similar functionality are possible, as they constitute valid *design choices* by which software architects make their design decisions. These decisions are precisely the soul of architectures, but they are often neglected during the architecting activity as they usually reside in the architect's mind in the form of tacit knowledge that is seldom captured and documented in a usable form. Furthermore, software architecture practice "encompasses significant decisions about

(i) the organization of a software system,
(ii) the selection of the structural elements and their interfaces by which a system is composed with its behavior as specified by the collaboration among those elements, and,
(iii) the composition of these elements into progressively larger subsystems" (from RUP).

For years, the generalized practice and research efforts have focused solely on the architecture representation itself. These practices have been for a long time exclusively aimed at representing and documenting the architecture of a system from different perspectives so-called architectural views. These views represent the interest of different stakeholders which are offered as a set of harmonized descriptions in a coherent and logical manner, and also used to communicate the architecture. IEEE standard 1471-2000 [3] provides a guide for describing the architecture of complex, software-intensive systems in terms of views and viewpoints, but it does not offer a detailed description of the rationale that guides the architecting process.

This paper describes the historic evolution of software architecture representation, and the role it can play, through a set of epiphanies which guides the reader from the initial architecture views to a new "decision view", expressing the need for capturing and using architectural design decisions and design rationale as first class entities. We summarize new activities that arise during the architecting process when design decisions are explicitly recorded and documented, and illustrate how this architectural knowledge constitutes a new crosscutting view that overlaps the information described by other views.

2. 1st Epiphany: Architectural Representation

Before 1995 and prior to the notion of architecture view, software designers did architecting, but the demand for large complex systems brought new design challenges. The intrinsic complexity of such systems, with different structures entangled in different levels of abstractions was organized into a set of architecture views that tried to describe the system from different perspectives, according to different users' needs. As a result, architecture views where proposed by Kruchten [4] in his "4+1" view model to provide a blueprint of the system from different angles. The set of views included in the "4+1" view model shown in Figure 1, was used by many Rational Software consultants in large industrial projects as part of the Rational Unified Process (RUP) approach. Four views are used to describe the design concerns of different stakeholders, plus a use-case view (the +1), that overlaps the others, and relates the design to its context and its business goals.

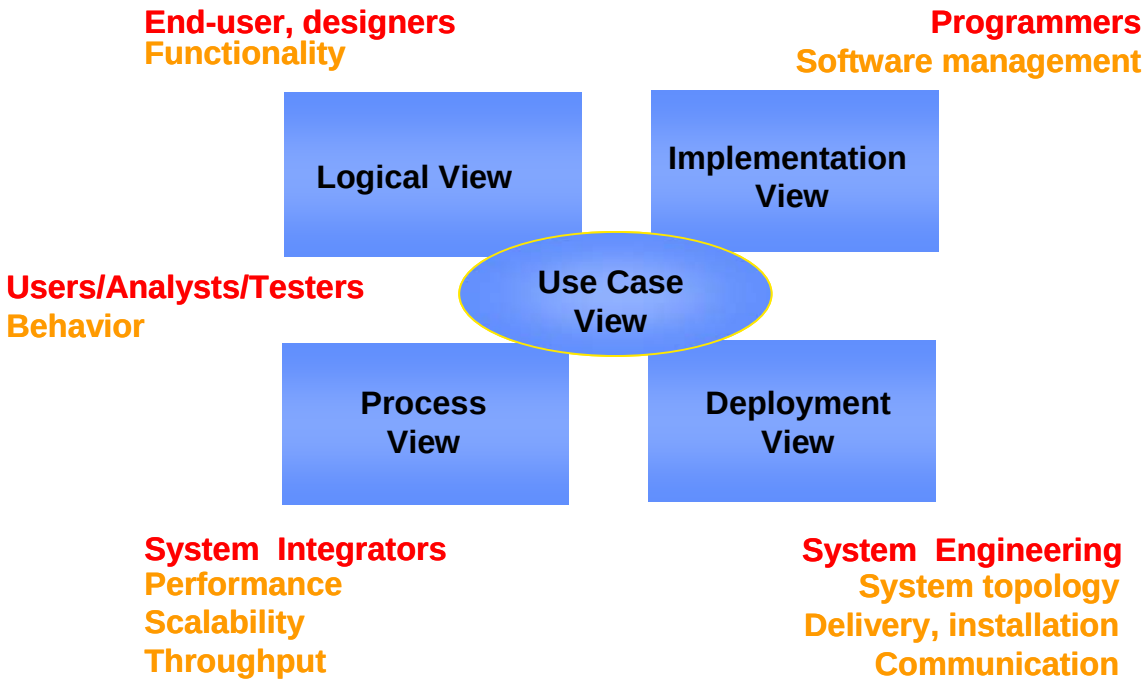

Figure 1: The "4+1" architecture view model [4]

Similarly, Siemens Corporation developed the Siemens Four-Views (S4V) method [5] based on best architectural practices for industrial systems and aimed to separate engineering concerns that reduce the complexity of the design task [6]. The goal of the proposed views is to help architects identify all influencing factors that can be used to find out the key architectural

challenges and to develop design strategies for solving the issues by applying one or more views. In such context, design decisions (i.e., a strategy which is applied to a particular view) are evaluated according to constraints or dependencies on other decisions. The Software Engineering Institute (SEI) proposes a classification based on views and viewtypes [7] and highlights the importance for documenting design decisions, but they give no details on how to do this nor the definition of adequate processes for capturing and documenting. Rozanski and Woods [8] clarify the most important architectural aspects or elements relevant for the stakeholders on information systems architecture, by defining up to six viewpoints. At this time, architecture research focused on the description and modeling of design and there was little agreement on notations for the representation of architecture.

2nd Epiphany: Architectural Design

The period between 1996 and 2006 brought complementary techniques in the form of architectural methods, many of them derived from well established industry practices. Methods like RUP (IBM), BAPO/CAFR (Philips), S4V (Siemens), ASC (Nokia), ATAM, SAMM and ADD (SEI), among others, are now mature design and evaluation practices to analyze, synthesize, and valuate modern software architectures. In some cases, they are backed by architectural description languages, assessment methods, and stakeholders-focused decision making procedures. Since many of the design methods were developed independently [6], they exhibit certain similarities and differences motivated by the different nature, purpose, application domain, or the size of the organization for which they were developed. In essence, they cover essential phases of the architecting activity but are performed in a different ways. Common to some of these methods is the use of design decisions that are evaluated during the construction of the architecture. These decisions are elicited by groups of stakeholders, under the guide of architects, but the ultimate decision makers are (a small group of-often a single person) architects. Unfortunately, design decisions and their rationale are still not considered as first-class entities as they lack a explicit representation. As a result, software architects cannot revisit or communicate the decisions made, which in most cases vaporize forever.

3. Reasons for Design Rationale

Rus and Lindvall wrote in 2002 that "*The major problem with intellectual capital is that it has legs and walks home every day*" [9]. Current software organizations suffer the loss of this intellectual capital when experts leave. The same happens in software architecture when the reasoning required for understating a particular system is unavailable and has not been explicitly documented. In 2004, Jan Bosch stated that "*we do not view a software architecture as a set of components and connectors, but rather as the composition of a set of architectural design decisions*" [10]. The lack of first-class representation of design rationale in current architecture view models brought the need to include decisions as first-class citizens that should be embodied within the traditional architecture documentation.

There are several benefits of using design rationales in architecture as a mean to explain *why* a particular design choice is made or to know which design alternatives have been evaluated before the right or the optimal design choices are made. Benefits can be achieved in the medium and long term, as architecture recovering processes - mostly used to retrieve the decisions when design, documentation, or even the creators of the architecture are no longer available – are avoided. In other cases, the natural evolution of a software system forces previous design decisions to be replaced by new ones. Hence, maintaining and managing this architectural knowledge requires a continuous attention to keep the changes in the code and the design aligned with the decisions, and to use these to bridge the software architecture gap.

It is in this new context when Perry and Wolf's old ideas [2] become relevant for upgrading the concept of software architecture by explicitly adding the design decisions that motivate the creation of software designs. Together with design patterns and assumptions, design decisions are a subset of the overall architectural knowledge (AK) that is produced during the development of architecture. Most of the tacit knowledge that remains hidden in the mind of the architects should be made explicit and transferable into a useful form for further use, easing the execution of distributed and collective decision-making processes. The formula Architecture Knowledge = Design Decisions + Design, recently proposed by Kruchten *et al*. [11], modernizes Perry and Wolf's [2] and considers design decisions part of the architecture.

4. 3rd Epiphany: Architectural Design Decisions

Architecture decisions are seldom documented in a rigorous manner, and explicitly documenting the key design decisions is a pretty rare case which can be only justified on political and economic reasons or even fear. Hence, our 3rd epiphany brings the need to deal with the representation, capture, management, and documentation of the design decisions that are made during architecting.

Active research during 2004-2008 has produced a significant number of approaches for representing and capturing architectural design decisions, as well as the definition of new roles and activities for supporting the creation and use of this AK. Several approaches have proposed template lists of attributes [11, 12, 13] to describe and represent design decisions as first class entities. The work described in [11] emphasizes the need for categorizing different types of dependencies between such decisions as valuable and complementary information to capture useful traces that can be used, for instance, during maintenance to estimate the impact when a decision is added, removed, or changed. Additionally, the research described in [13] advocates for the use of flexible approaches that employ mandatory and optional attributes for knowledge capture that can be tailored to each specific organization. Others have proposed ontologies [11] to formalize tacit knowledge and make visible the relationships between the decisions and other artifacts of the software lifecycle. The field of product family engineering or product lines has produced a large amount of work about specification, modeling and automation of design decisions applied to the description and selection of both common and specific elements in the

product line [16]; in this particular case knowledge is codified in an operational manner, as derivation processes are automated.

5. New Architecting Activities

It is well recognized that design decisions have a great impact both in software architecture and software engineering, and several authors have recently contributed with models, methods, and tools that encourage the use of design decisions [14]. Because architecture modeling is not an isolated process from the decision-making activity, new processes have to be carried out in parallel with typical modeling tasks. Hence, architecting is highly impacted by these new activities that deal with the creation and use of design decisions.

Thus, software architects, as decision makers, have to assume new roles in the form of knowledge producers and consumers in a social process, and perform a variety of new activities. These two aspects are shown in Figure 2 (inspired by technical report by P. Lago and P. Avgeriou of the First SHARK workshop in 2006) to articulate the decisions made and the architecture resultant of these decisions. For instance, architects make new decisions ("create") which lead to a particular architecture. In this phase, decisions are captured and characterized in a usable form and linked to design artifacts. Once a first version of the architecture is created, the design can be communicated to other stakeholders ("share"), and perform for instance, a review process of the status of the architecture. During maintenance, decisions may become relevant for the current architecting team for evaluation and to provide recommendations in order to determine if the decisions made were right or wrong. Because the architecture is continuously evaluated, assessment procedures can be carried out at different stages of the architecture development process (when decisions are made for the first time or after). Also, less expert architects can learn from decisions made by other; if wrong decisions are detected, these must be fixed or replaced by new decisions, and the architecture should be modified accordingly. As a result, a perfect alignment between decisions and design can be achieved.

Additional sub-activities refine the main ones shown in Figure 2, but our aim is to explain that parallel complementary activities related to the reasoning process influence directly the architecture modeling tasks. We justify the separation between knowledge producers and knowledge consumers based on the distinction between architecting for the first time and the maintenance of the architecture over time.

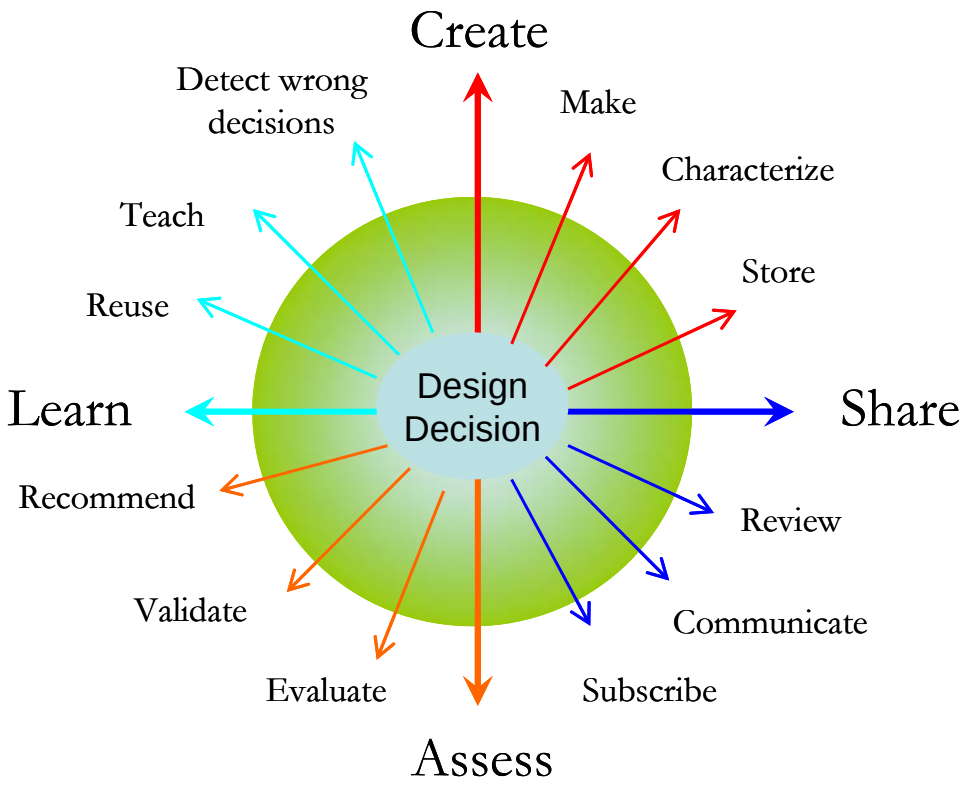

Figure 2: Example of activities and sub-activities involved in creation and use of design decisions and design rationale. Two main roles "create-share" and "assess-learn" organize the smaller activities.

6. Impact and Use

Our 3rd epiphany introduced a strong impact in current architecting practices as the value of capturing and using design decisions has already been reported in a few empirical studies that provide some results on the following aspects:

(i) the perceived value of designs decisions and design rationale for different kinds of stakeholders, as different items for representing and recording the information of design decisions may not have the same importance for all stakeholders [17]. Hence, we should decide which type of knowledge would better fit each type of users,

(ii) the effort in capturing decisions during the early development stages pays off in later maintenance and evolution phases, thus expecting return of the investment when decisions are captured for the first time [18]. The experiences described there also

highlight the benefits of using specific tool support for capturing, managing, and documenting architectural design decisions.

Another visible consequence refers to the information documented by means of the traditional views that is described in the IEEE std. 1471-2000 *Recommended practice for architectural description of software-intensive systems* [3]. Its successor, known as ISO/IEC 42010 and currently under review, expands it with architectural knowledge, including among others concepts: concerns, design decisions and rationale,.

A complementary perspective in which decisions are entangled with design for each architectural view, leads us to think about a "new" architectural view, called the "decision view" [15]. This new perspective extends the traditional views by superimposing the design rationale that underlies and motivates the selection of concrete design options. Figure 3 depicts a graphical sketch of the "decision view", in which design decisions are attached in the "4+1" view model.

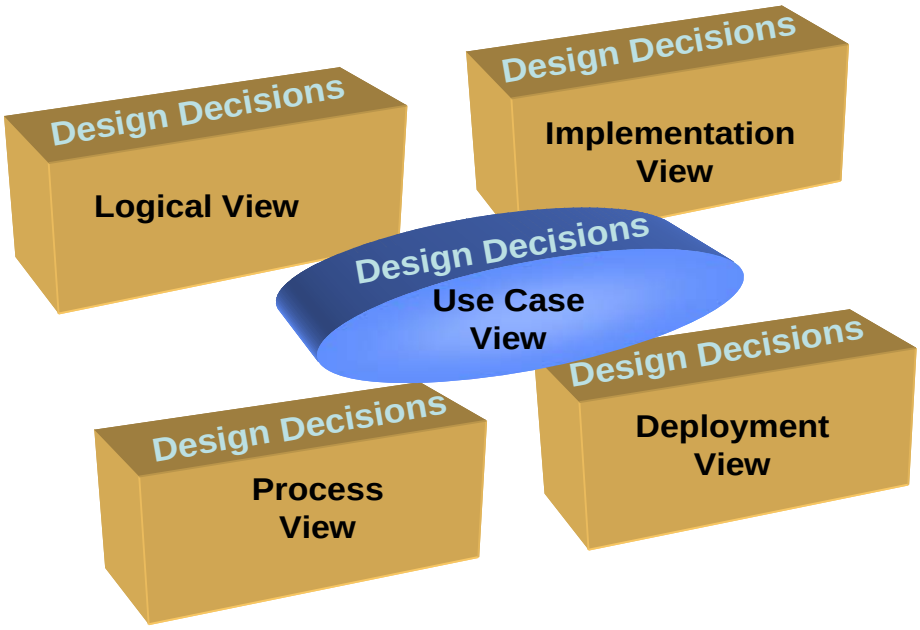

Figure 3: The "decision view" view embedded in the "4+1" view model

6.1 The Texture of a "Decision View"

The traditional representation of architectures in terms of views and viewpoints varies when decisions have to be described. Hence, architects interested in capturing the decisions and rationale should know *how* a decision view can be built, that is, the texture of decisions and how these are represented. As a first approach, we can refer to the classical methods for architectural assessment; most of them rely on the development of scenarios, their projection against several candidate architectures, and addition of information to the architectural components; later this information is aggregated and evaluated for each candidate architecture. We presented a case of architectural assessment and definition of design decisions on a product-line architecture for medical equipment where the decisions relate to the economical impact of changes of each architectural component [17]. The authors focused on the economical attributes of each component in the implementation view (from the 4+1 model), and their decision view is composed by the decisions, rationale and actual data on the architectural components.

Focusing on the capture and representation of decisions itself, as a guide to help architects to document the decisions in their architectures, we propose the following actions:

(i) decide which information items are needed for each design decision (e.g.: name of the decision, description, rationale, pros and cons, status, category, etc.). Then, decide which representation system will better ease the recording and organization of the decisions (i.e.: templates, ontologies). A strategy to capture the items (e.g.: codification, personalization, hybrid) should be also selected,
(ii) for each decision, define links to the requirements that motivate it,
(iii) in case of alternative decisions that need to be evaluated, provide mechanisms to change the status of the decision (e.g.: approved, rejected, obsolete) and category (e.g.: from alternative decision to a main decision),
(iv) in case a decision may depend on previous ones, define these relationships in order to support internal traceability among them,
(v) once a set of significant set of decisions are made, link them to the architecture that results of such decisions. These links provide the connection to traditional architecture views,
(vi) after all the decisions are made and captured, share them by means of communication and documentation mechanisms.

Extra items and functionality can be added to this list (e.g.: support the evolution of decisions), but we believe these are enough for a quick start in capturing design decisions and its underpinning rationale alongside with their architectures.

6.2 Challenges and Benefits

The explicit capture and documentation of design decisions will bring new challenges that in most cases can be seen as benefits derived from the use of the decisions in architecture development. Some of the expected challenges/benefits are:

(i) decisions enhance traceability between software engineering artifacts produced across the software life-cycle; forward and backward traces facilitate the understanding of the root causes of changes and help in better estimating change impact analysis,

(ii) capturing the dependencies between decisions supports impact analysis when a decision is added, modified, or removed,

(iii) documented decisions facilitate the general understanding of a system, which is particularly useful during staff turnover,

(iv) documented decisions facilitate knowledge sharing and assessment processes because users can easily review the rationale of past decisions,

(v) learning activities can use previous knowledge for assessing novice software architects in their professional careers,

(vi) leveraging tacit architectural knowledge into formal documentation requires understanding and performing many of the activities described in Figure 2.

The adoption barrier for capturing design rationale can be high because of the intrusiveness the new activities stated in Figure 2. Therefore, the overhead required during the creation of these decisions should pay off during maintenance, as key design decisions avoid the need to reverse architecture descriptions from code, particularly in staff turnover situations, or rapid software evolution. Long-term benefits and a reduction of maintenance costs (e.g., architecture recovery) are expected to motivate users to capture the design rationale, in particular in successive iterations of the system evolution [18]. Hence, the broad impact for capturing and using architecturally significant design decisions affects not only the evolution of designs but also the evolution and maintenance of the decisions base itself. This issue often emerges during reviews, where major changes affect the design designs. Like other key activities, recording the history of decisions is another challenge that should be treated in depth.

7. Sidebar: Tools Supporting Design Rationale

As pointed by Dutoit and his colleagues [14], "*the design rationale (DR) movement began with Rittel's IBIS (Issue-Based Information System)*" early in the 1970s for supporting Design Rationale in general. The IBIS approach and its successor gIBIS, were applied to large-scale projects in the 1970s and 1980s and they include some basic items for supporting the design rationale and to discuss controversial questions that arise in design. On the basis of Rittel's approach, other tools like: PHI (Procedural Hierarchy of Issues), QOC (Questions, Options, and Criteria), and DRL (Design Representation Language), appeared in the field of Design Rationale as extensions of the IBIS tool. Other tools (SCRAM, C-ReCS, SEURAT, Sysiphus or DRIMER)

developed between 1992 and 2004 [14], provide simple solutions to manipulate knowledge and record decisions for a broad number of software engineering processes. Since 2005, active research has produced a number of tools supporting design rationale in software architecture. In this research we have identified five representative research prototype tools aimed for: *capturing*, *using*, *managing*, and *documenting* architectural design decisions.

Archium (http://www.archium.net) is an extension of the Java language that provides traceability among a wide range of concepts such as requirements, decisions, architecture descriptions, and implementation artifacts, which are maintained during the system life cycle. The Archium tool suite consists of a compiler, a run-time platform, and a visualization tool. The compiler turns the Archium source files into executable models for the run-time platform. The visualization tool uses the run-time platform to visualize and make accessible the architectural knowledge.

The Architecture Rationale and Element Linkage (**AREL** - http://www.ict.swin.edu.au/personal/atang/AREL-Tool.zip) is a UML-based tool to assist architects to create and document architectural design with a focus on architectural decisions and design rationale. AREL captures three types of AK: *design concerns, design decisions* and *design outcomes*. These knowledge entities are represented by standard UML entities and they are linked to show the relationships between them.

The Process-based Architecture Knowledge Management Environment (**PAKME** - http://193.1.97.13:8080/) is a web-based tool built on top of the Hipergate open source groupware platform, aimed at providing collaborative knowledge management support for the software architecture process. PAKME's features can be categorized into four AK management services: knowledge acquisition, knowledge maintenance, knowledge retrieval, and knowledge presentation.

The Architecture Design Decision Support (**ADDSS -** http://triana.escet.urjc.es/ADDSS) tool is an ongoing research web-based prototype which captures design decisions using a template list of mandatory and optional attributes that supports a combined strategy (codification and personalization). Decisions are related to requirements and architectures. The tool provides an automatic reporting system that produces documents containing the decisions made for a given architecture, the trace relationships from decisions to requirements and architectures and also between decisions. In addition, ADDSS users can navigate and visualize the architectures and the decisions showing the evolution of the system over time.

The **Knowledge Architect** (http://search.cs.rug.nl/griffin) is a tool suite for capturing, managing, and sharing AK that uses a server and an AK repository, which is accessed by several plug-in clients: a Word client to capture and manage AK in MS Word documents, a second one captures and manages the AK of quantitative architectural analysis models using MS Excel, a

third client, the Knowledge Architect Explorer, is a visualization tool that supports the analysis of the captured AK. This tool allows for the exploration of the AK by searching and navigating through the web of traceability links among the knowledge entities.

[end of side bar]

8. Conclusion

The perception by the software architecture community that architectural design decisions are an intangible asset that is difficult to capture and communicate is now changing as a result of recent research that is leading to a new perspective or "view", in the IEEE 1471 sense, to describe rationale and architectural knowledge. The traditional gap between different artifacts of the software engineering activity has shown the need to capture and represent design decisions and their underlying rationale in an effective and precise manner for later use in order to avoid knowledge vaporization.

We also believe that key design decisions should be recorded and documented, in contrast to the effort required to capture and maintain all the micro-decisions that happen along the life of a software system. One of the adoption barriers for capturing design decisions is the intrusiveness of many of the processes described in Figure 2, as they are not fully integrated into current software engineering activities. Therefore, tools like those mentioned in the sidebar, need to be improved, adapted, or better integrated to avoid duplicate efforts in capturing the design decisions, and to facilitate the gradual introduction of new activities that deal with design rationale, some of which are related to the distributed team decision making process.

We have observed that there is a small distinction between software requirements or the description of a well-known design pattern and the explicit representation of a design decision. In many cases, a design decision constitutes a replica of the requirement that motivated such decision, and no distinction is made as a further refinement of the user needs. As a result, the effort in capturing such decisions is considered duplicated, because users of the aforementioned tools often record the same data. Therefore, appropriate mechanisms should be provided to avoid those cases where the same information is recorded as well as to streamline the capturing effort. These mechanisms will be based on stronger tracing and duplication-detection techniques.

The key goal of this research is to highlight the importance and the impact of design rationale in software architecture activities in particular, and in software engineering from a broader perspective. What will a 4^{th} epiphany bring? Despite the challenges of capturing the design rationale, the introduction of documented design decisions will bring better ways to build and understand our software systems, and software architects and developers will perceive the utility of considering decisions as first-class entities as well as to pursuit a better integration with other

software engineering artifacts. Design decisions and design rationale will hopefully be recognized in the upcoming ISO/IEC 42010 standard.

References

[1] P. Kruchten, H. Obbink, and J. Stafford, “The Past, Present, and Future of Software Architecture,” *IEEE Software* vol. 23, no 2, pp. 22-30, 2006.

[2] D.E. Perry, and A. L. Wolf, "Foundations for the Study of Software Architecture," *ACM Software Eng. Notes* vol. 17, no. 4, pp. 40–52, 1992.

[3] *IEEE 1471:2000, Recommended Practice for Architectural Description of Software-Intensive Systems*, Los Alamitos: IEEE, 2000.

[4] P. Kruchten, "The 4+1 View Model of Architecture," *IEEE Software*, vol. 12, no. 6 , pp. 45–50, 1995.

[5] C. Hofmeister, R. Nord, and D. Soni, *Applied Software Architecture*, Boston: Addison-Wesley, 1999.

[6] C. Hofmeister, P. Kruchten, R. Nord, H. Obbink, A. Ran, and P. America,, “A General Model of Software Architecture Design Derived from Five Industrial Approaches,” *Journal of Systems and Software*, vol. 80, no. 1, pp. 106-126, 2007.

[7] P. Clements, et al. *Documenting Software Architectures: Views and Beyond*, Boston: Addison-Wesley, 2002.

[8] N. Rozanski and E. Woods, *Software Systems Architecture*, Boston: Addison-Wesley, 2005.

[9] I. Rus and M. Linvall, “Knowledge Management in Software Engineering,” *IEEE Software*, vol. 19, no 3, pp. 26-38, 2002.

[10] J. Bosch, "Software Architecture: The Next Step," in. *Proc. 1st European Workshop Software Architecture* (EWSA 04), LNCS 3047, pp. 194–199, Springer-Verlag, 2004.

[11] P. Kruchten, P., Lago, and H. van Vliet, “Building up and Reasoning About Architectural Knowledge,” in *Proc. QoSA2006* , LNCS 4214, pp. 43-58, Springer-Verlag, 2006.

[12] J. Tyree and A. Akerman, “Architecture Decisions: Demystifying Architecture,” *IEEE Software*, vol. 22, no 2, pp 19-27, 2005.

[13] R. Capilla, F. Nava, and J. C. Dueñas, “Modeling and Documenting the Evolution of Architectural Design Decisions,” in *Proc. of the 2nd Workshop on Sharing and Reusing architectural Knowledge Architecture, Rationale, and Design Intent*, IEEE CS, 2007.

[14] A. H. Dutoit, R. McCall, I. Mistrik, and B. Paech (Eds.) *Rationale Management in Software Engineering.* Berlin: Springer-Verlag, 2006.

[15] J. C. Dueñas and R. Capilla, “The Decision View of Software Architecture,” in *Proc. of the 2nd European Workshop on Software Architecture* (EWSA 2005) LNCS 3047, pp. 222-230, Springer-Verlag, 2005.

[16] T. Käkölä and J.C. Dueñas (eds), *Software product lines - Research issues in Engineering and Management*, Springer-Verlag, 2006.

[17] D. Falessi, R. Capilla, and G. Cantone, “A Valued-Based Approach for Documenting Design Decisions Rationale: A Replicated Experiment”. *Proceedings of the 3rd international workshop on Sharing and reusing architectural knowledge* (SHARK’08), pp. 63-70, ACM 2008.

[18] R. Capilla, F. Nava, and R. Carrillo, “Effort Estimation in Capturing Architectural Knowledge,” *Proceedings of the 23rd IEEE/ACM International Conference Automated Software Engineering,* L’Aquila, Italy, September 15-19, 2008.