CUSTOMIZING THE CAPTURE OF SOFTWARE ARCHITECTURAL DESIGN DECISIONS

Larix Lee, Philippe Kruchten

University of British Columbia

ABSTRACT

Significant challenges arise when capturing architectural knowledge and design decisions for a software project, resulting in high capturing effort and few captured decisions. These challenges stem from the fact that software development is a group activity that involves different situations, goals, methodologies and needs of the software development organization. A project's documentation processes should be flexible to suit the needs and goals of an organization and allow multiple approaches. We propose three approaches for capturing design decisions to increase the knowledge retention and decision capture rates in an organization: 1) formal elicitation; 2) lightweight top-down capture; and 3) lightweight bottom-up capture. Customizable capturing methods are suggested to implement the approaches and a tool is created for each approach. Industry feedback and preliminary findings show the feasibility of the proposed approaches.

Index Terms— software architecture, design decisions, architectural knowledge, decision capture, design rationale

1. INTRODUCTION

The definition of software architecture is evolving and increasingly recognizing the role that architectural design decisions play in the architecture itself. As defined by various groups, software architecture is composed of a collection of architectural design decisions [1, 2, 5, 6]. For example, the RUP defines software architecture as the *set of significant decisions* about the organization, structure, style, and behaviour of a software system [6]. Bosch recommends representing software architecture as "the composition [of] a *set of architectural design decisions* concerning, among others, the domain models, architectural solutions, variation points, features and usage scenarios needed to satisfy the requirements" [1]. These definitions describe software architectural design decisions as cross-cutting, governing the form or behaviour of a software system. For example, choosing J2EE over .NET is a design decision.

As we are now representing software architecture as a set of decisions, we need to find the means to capture and document these decisions with sufficient detail so that an accurate architectural representation can be made. However, capturing software architectural design decisions differs from current capturing methods in practice today (like UML or entity-relationship diagrams) because architectural design decisions are fundamentally knowledge-based.

An important decision a software development organization may need to make is to determine how they will document the architectural design decisions for a project. The organization can capture design decisions from without (top-down) or from within (bottom-up), depending on what would be more beneficial for the organization's particular situation. The organization can also choose when they would prefer to capture the majority of the decisions of a project's implementation. Another choice is how much decision documentation an organization deems to be sufficient.

The main idea is to find the right low-impact capturing method for an organization to assist in capturing their design decisions to reduce effort. The research goal is to find a systematic way to adapt the capturing process to the needs of each organization, not finding a winning formula. We propose that a software development organization could use at least one of three approaches to improve the capture of architectural design decisions. The purpose of this paper is to describe the three approaches and their implementation.

The paper is structured as follows: Section 2 provides background into architectural knowledge and design decisions. Sections 3 and 4 describe three approaches and methods to capture software architectural design decisions. Section 5 describes three tools that implement the proposed approaches, and sections 6 and 7 discuss industry feedback and preliminary findings. Section 8 summarizes and concludes our discussion on tailoring the capture process.

2. DECISION CAPTURE PROBLEM

Being knowledge-based, design decisions have also inherited the difficulties surrounding knowledge capture. There are two particularly problematic areas. One area deals with the type of knowledge being captured and how the knowledge can be accessed. The other area involves assessing an organization's need for knowledge.

978-1-4244-1643-1/08/$25 ©2008 IEEE

2.1. Architectural knowledge

Knowledge can be categorized into three levels [10]: tacit, documented, and formal. Documented knowledge is knowledge that is captured in some form outside the minds of people, similar to the content in a diary. Formalized knowledge is a particular case where the knowledge is documented and structured in an organized fashion, like a dictionary, so that it can be exploited using other ways. Tacit knowledge is acquired from experience and is difficult to express. It remains in the mind, but it can be forgotten.

Applying this concept to architectural knowledge suggests three ways of access. Architectural knowledge can stay tacit and remain undocumented in the person's mind. Questions raised in the design are directed to various people within the organization in hope that someone may know something about it. The second way involves quickly documenting the architecture in a vague manner as a reminder to the person who captured the knowledge. Here, knowledge is documented, but relies on the interpretation by the person who has the knowledge. The third way is to explicitly document self-contained knowledge that does not require data interpretation.

2.2. Knowledge needs

Depending on the type of project being developed, the importance and method of capturing architectural knowledge can vary. If the projects are small, such as websites or utility tools, the effort needed for architectural knowledge capture may be unjustified for the small size, thereby making documentation unnecessary. For large or high-risk projects, like safety-critical systems, higher stakes require careful planning, implementation and maintenance.

Who the knowledge is for is another factor to consider. A large company with many projects with high turnover rates would prefer to increase the corporate knowledge of the software systems being produced to improve code design and maintainability. On the other hand, an emphasis on individual knowledge decreases time spent on knowledge capture and increases productivity because the designer or architect is not required to share the knowledge learned.

A compromise between the two extremes can benefit organizations that do not like either case. In some cases increasing corporate knowledge is beneficial and in other cases providing better opportunities for individuals to learn and keep track of knowledge is preferred. If the procedure to capture corporate knowledge can be broken down into smaller, terminable steps that enable self-contained capturing mechanisms for individual benefit, then organizations can find the right balance.

3. SOLVING THE CAPTURE PROBLEM

As the benefits of using architectural design decisions ultimately rely on the acquisition of decisions, it is therefore necessary to have effective means of decision capture. We propose a three-pronged approach to decision capturing. These three approaches are formal elicitation, lightweight top-down capture, and lightweight bottom-up capture.

3.1. Formal elicitation

Formal elicitation of software architectural design decisions is the method of gathering software decisions in an explicit and structured manner, where this is normally performed in several long sessions devoted for this purpose. There are many existing decision capture methods that implement the formal approach and they follow the formal modeling styles that capture architectural knowledge and design reasoning, many of which are described in [3, 8, 9]. Figure 1 illustrates the formal elicitation approach generalized from those methods. We will briefly outline the main ideas.

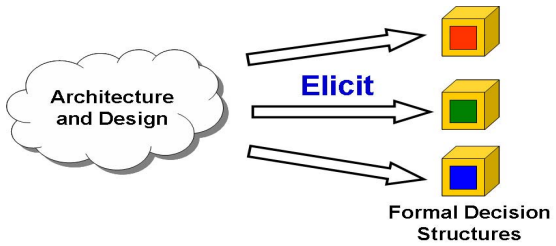

Figure 1: Generalization of formal elicitation

In this approach, decisions are elicited upfront, with emphasis on gathering detailed information on decisions. Details include the issues, alternatives, choices, and rationale that were present when the decision was made. These details help make the captured design decisions more self-expressive so that someone new to the software system can quickly understand the nature of the design through the decisions' context. The focus on gathering detailed information allows more accurate modeling of a designer's decision processes, and hence the types of information gathered through formal elicitation are considered to be the fundamental structure for modeling, manipulating and browsing design decisions. Decisions created at the end of the elicitation process are well structured and can follow one of many design rationale and decision representation models mentioned above.

3.2. Lightweight top-down capture

To complement the formal capturing methods mentioned above, we propose a new lightweight capturing approach for software architectural design decisions. This approach focuses on the early design phases of a software project and attempts to support software architects and designers in performing their activities. The term "lightweight" refers to the ability to capture incremental and incomplete knowledge. A method that implements this splits the formal elicitation approach into three steps: flag, filter, and form. This method is illustrated in Figure 2.

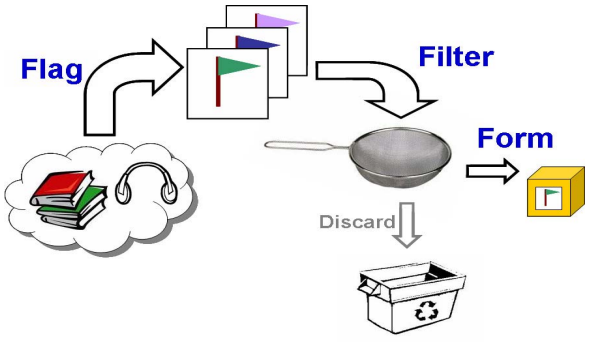

Figure 2: The flag, filter, and form steps

3.2.1. Flagging

Flagging is the capture of candidate decisions from the source in which they are found. Candidate decisions are ones that are considered, but not necessary for a design. A decision candidate can be captured by using a reference marker, which briefly describes the essence of the decision and points to the source where it is found. Flagging can be performed with little worry over the immediate relevancy or priority of the decisions as the sorting tasks can be performed at a later point in time during the filtering step.

3.2.2. Filtering

After a period of time, the accumulated decision references would require some sifting to identify which decision candidates are still applicable for the project. This promotes periodic cleanup of the list of decision references to reduce the amount of obsolete decisions in the final decision documentation. Identified relevant decision candidates are considered for formal decision structuring.

3.2.3. Forming

The purpose of the forming step is to fill out the details and contextual information of the relevant decisions identified during the filtering step. A formal decision representation model is used to structure the decision and its attributes in an organized and accessible manner so that the captured decisions can be recalled, analyzed, or manipulated at a later point in time. For example, one type of decision model entails documenting the epitome, rationale, scope, state, categories, author, and the creation time [7].

3.3. Lightweight bottom-up capture

Viewing lightweight decision capture from the software programmer's perspective, we propose a second new capturing method that supplements both the formal capturing methods and the lightweight top-down method described above. The goal of the lightweight bottom-up approach is to capture architectural decisions that are documented within the many artifacts generated during software development. Again, "lightweight" refers to the ability to capture incremental and incomplete knowledge. To demonstrate this approach, we suggest a capturing method that has two steps: tag and form. This method is similar to the three-step capturing method described in section 3.2, but is tailored to better suit the needs of programmers and maintainers. Figure 3 illustrates this method as applied to software source code.

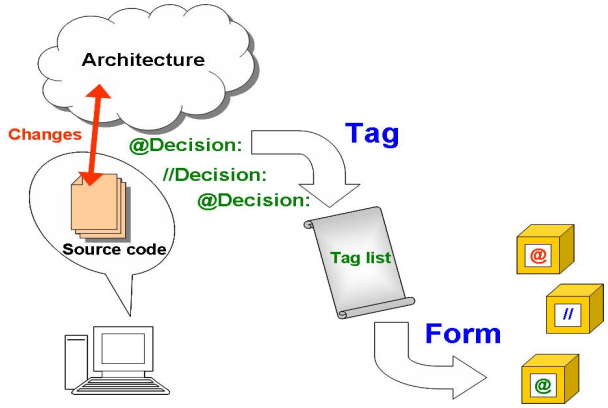

Figure 3: The tag and form steps applied to source code

3.3.1. Tagging

Tagging is "flagging" of decisions that are reflected in the various design artifacts generated throughout software development. These artifacts include software code, models, requirements, or text-documents that describe the software design. Moreover, the artifacts should be accessible and be uniquely referenced for long-term traceability.

In the context of software code, tagging is a common term to describe the act of placing identifiers within source code or on a collection of files to store specific information for future reference. For decision capture, tagging refers to placing identifiers within the design artifacts to document design decisions made that deals with the particular design section represented in the artifact. A related work used tagging and code commenting as "waypoints" to document thought flow and code navigation [11].

3.3.2. Forming

The decisions tagged within source code are typically terse and would not contain the level of detail necessary for formal decision representation; therefore, a separate step is needed to form the decision using supplementary details. This step can be performed during peer code review to encourage knowledge dissemination and increase architectural awareness, or this can be done semi-transparently through routine code commit comments.

4. CUSTOMIZED DECISION CAPTURE

The three decision capturing approaches allow organizations to choose the right approach for their needs. Each approach addresses a particular perspective of decision capture, and the choice of which approach will depend on the situation. It is possible that all three capturing approaches can be performed simultaneously if the organization is willing.

4.1. Comparing the three approaches

The goal of all three approaches is the same: to create formalized decision entities that can be manipulated and analyzed. Figures 1-3 illustrates this goal by showing the creation of decision "blocks" at the end of each method. In essence, the two lightweight approaches are the result of breaking down formal elicitation to multiple smaller steps, analogous to the principle of transitivity. By completing all the steps of a lightweight approach, the resulting formal decision is effectively equivalent to the decision if it was captured using the formal elicitation approach. However, the formal and lightweight approaches are not truly transitive, as the resulting decision sets from a lightweight capturing approach may contain additional information that would have otherwise been lost if the decisions were created through the formal elicitation approach. These include backwards traceability, information sources, background context, and discussion traces.

On the other hand, decisions created through formal elicitation may contain more relevant information, as some design details can be forgotten or lost between different steps. The differences between the approaches suggest that there are advantages where a particular approach is more useful for certain situations.

Formal elicitation is useful in situations where the decisions are made with some level of confidence. This is usually the case during technical design discussions where bursts of decisions are generated in response to the issues at hand. Formal elicitation is also useful when the decisions are already made and readily available, such as during post-implementation reverse engineering, design comprehension, and documentation. However, the main concern of using the formal elicitation approach is that it requires significant effort to enumerate the decisions someone made when designing a particular system. The result is a significant upfront cost in which interested participants become discouraged by the amount of effort being expended. The results of a survey support this view [12].

The two lightweight approaches are designed to facilitate and support the design activities of the software architects, designers, and the rest of the software development organization. The focus on lightweight, incremental capture of design decisions is that it assists software architects and designers by reducing the impact of the decision capturing process on their design activities.

Top-down capture of design decisions benefit mostly when software architecture is in its early design stages, where decisions are vague and subject to change. Here, detailed capture is not possible or warranted. However ambiguous, early-stage decisions are important to capture, as they make up the foundations of the design and determines the path of progression for the design.

The bottom-up approach is suited for situations where architectural decisions are made during a project's implementation and maintenance phases. In these cases, capturing the technical details of an issue may involve referencing numerous external articles, from technical service bulletins to discussion threads and to internal source code. Since a technical issue can stem from a particular section of code or architectural diagram, decisions can be made and captured as close as possible to the troublesome area. The result of using a bottom-up decision capture approach is that it enables more precise insight into the issues and the resulting decision can also be easily referenced throughout the lifetime of the design artifact because it is a part of the product.

4.2. Customizing each method

Although the three approaches can be viewed as a means to an end (in that the final result is the creation of a formal decision entity to represent the architectural knowledge), there is no restriction as to what the final form of the captured knowledge should be for an organization. In section 2.2, we mentioned about the knowledge needs of an organization. Each capture approach satisfies a certain design process perspective and each method implementation can adapt to varying capture goals.

For some organizations, using a portion of a lightweight method suffices; just capturing decision references in its unrefined state is sufficient enough for them as architectural decision documentation. The underlying implication is that the organization would rely on the people involved to provide additional information or interpretation of the data. This concept is related to the work of contribution structures [4], as there is a need for organizations to maintain authorship traceability for the captured decision references. Likewise, capturing architectural knowledge within the source code without enforcing the formalized decision representation may be just as acceptable for an organization. The objective is to adapt the capturing method to suit the needs of an organization without imposing additional work.

5. TOOL SUPPORT

We created a tool for each of the three decision capture approaches to demonstrate the feasibility and assess the benefits of the approaches. For the formal elicitation approach, we decided to capture design decisions in a structured form described in [8], such as the description, rationale, scope, and state. Decision relationships are also included. The tool is designed to be used in highly-technical architectural discussion sessions where decisions are created or modified in batches in response to a technical discussion. The tool focuses on the utility of capturing design decisions, and it allows users to browse and modify the collection of decisions for a project. When creating a new decision, the user is shown a blank decision-attribute dialog where the user can fill in the details of the decision. The user would save the decision and append a change comment, then continue eliciting or browsing decisions. Figure 4 depicts the formal elicitation tool while a user is saving a decision.

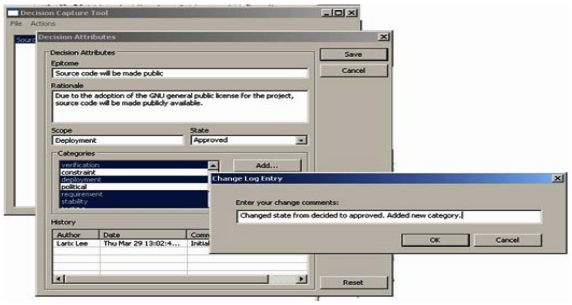

Figure 4: Formal elicitation tool with decision details dialog open for editing and committing

The top-down capture tool implemented the flag-filter-form method. Named "DecisionStickies", the tool attempts to model after sticky-notes. This tool, shown in Figure 5, is based on the way someone can write on and use sticky-notes as bookmarks for later information retrieval.

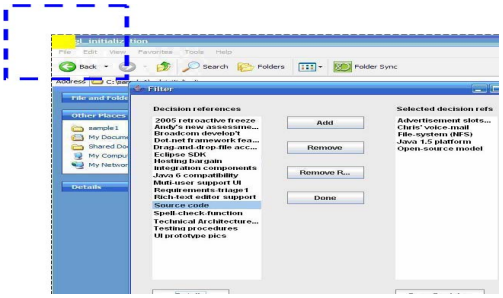

Figure 5: DecisionStickies on a typical desktop with filter dialog box opened. Dotted-box highlights main interface

A user can flag a decision reference by drag-and-dropping a file or an e-mail onto the tool and the user can then enter some quick information about the decision reference and continue with whatever the user is doing. To filter decisions, the user is shown a screen with two lists of decision references: available and selected. The references can be moved between the two lists and the user can form the decisions similar to the formal capture tool.

For a bottom-up capture tool, we focused on capturing decisions stored in software code, shown in Figure 6.

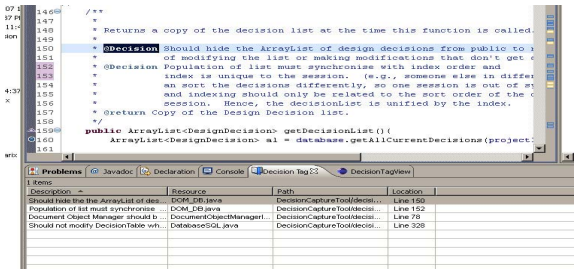

Figure 6: Eclipse plug-in decision tag view with decisions

We implemented an Eclipse plug-in to parse through all the code in the project's source files, identify all decision tags (denoted by an "@Decision" or "//Decision" comment), and display those tags in a "view" within Eclipse. Though not currently implemented yet, the tool can form decision entities in a way similar to the formal capture tool.

6. INDUSTRY FEEDBACK

We presented the tools to three industry participants representing separate software organizations. We were able to gain feedback regarding the tools and the capturing approaches the methods represent. During the initial contact, all three participants expressed the desire to capture their architectural knowledge and acknowledged that current capturing processes were insufficient for architectural knowledge. All three organizations used requirements documents and UML for their architectural documentation.

The first participant was hired to manage a project already underway in a small-medium game-development company based in North America. The participant would like to capture current design decisions and relay them to developers in Asia to reduce the amount of communication overhead. This participant has stated interest in the top-down capturing tool to assist in decision capture as the participant would like to learn and document decisions made before the project started, As well as keeping track of the decisions the participant has made already. The participant's past experiences with heavy documentation resulted in less motivation to document knowledge, so the choice of the lightweight top-down approach is appropriate.

The second participant represents a small software development organization that specializes in information and knowledge management. The participant has expressed a need to capture architectural knowledge of system being developed for future reuse, and the participant is actively capturing knowledge and background information on the project. The participant expressed enthusiasm for the formal and the top-down tools, but the bottom-up tool was not discussed in detail as the participant's project has already started. The participant did state that, the bottom-up approach is interesting and serves purpose.

The third participant is a software architect from a large organization that highly values documentation and established software development processes. The participant is involved in a large, multi-national project in its mature development stages and the organization would like to document decisions with us through meetings and design documents. The participant explained that they cannot apply the lightweight capture methods to their situation because most of the architectural design decisions have already been made and code implementation is well underway. Thus, the consensus to use the formal elicitation approach is appropriate for them in this post-design decision capture.

7. PRELIMINARY FINDINGS

The participants agreed to collaborate with our research by providing us with their project decisions. Due to time and resource constraints of the participants, we elicited decisions

from the participants, which is acceptable for a feasibility study on the approaches.

One decision set came from a mature project, and we were able to acquire around 40 decisions through two meetings dedicated to the decision capture and the concept and overview-requirements documents. Using the formal elicitation tool, we found that capturing decision rationale from documentation is difficult and we heavily leverage the discussion during the technical meetings for the architectural decisions and their rationale. Of the 12 documented decision relationships, three relationship types were documented (5 are the "enables" type, 5 are the "constrains" type, and 2 are the generic "is-related to" type). Defining relationships were difficult if we did not repeatedly ask whether this decision was related to another decision. We found that cross-referencing keywords and scope helped find relationships.

The decision set that came from the participant keen on capturing background information and knowledge provided a good opportunity to use the top-down capturing tool to acquire decisions. We received extensive background documentation, such as statement of work, requirements, e-mail, and other internal assessment documents. From the documentation alone, we captured 83 decision references in the first iteration of decision capture, of which 62 are selected after filtering. All 62 filtered references were formed into decisions. The 21 remaining references were either redundant or were irrelevant due to scope change mentioned in the documentation. The iteration spanned four weeks, averaging close to an hour per session with two sessions per week. The capture of decision rationale and relationships was less difficult than with the first decision set, likely due to the availability of background information.

Since all the participants have started implementing their projects, we did not get the opportunity to evaluate the bottom-up capture tool in industry. However, we used the bottom-up capture tool on our formal elicitation tool source code, which contains some decisions. The tool identified 14 decision tags; 8 were unique due to code reuse and common interfaces provided by design patterns.

8. CONCLUSION

We proposed three approaches of decision capture that organizations can use to customize their decision capture processes. We suggested methods that implement the approaches, and we created three tools using the methods described here. Furthermore, these methods can be further tailored to the needs of an organization. Initial feedback from participants supported the need for customizing the decision capturing process to each organization. Using the developed tools and decisions acquired from participants demonstrated the feasibility of the approaches. Continued research and decision dataset acquisition will be used to evaluate the exploitability of the design decisions and determine how we can continue to enhance the decision capturing processes by improving on the suggested methods.

ACKNOWLEDGEMENTS

We would like to thank Elaine Ting and Teresa Zhou for their research assistance. We appreciate the help of William Ha, who helped implement the DecisionStickies tool. We especially would like to thank the three study participants who volunteered their time to contribute to our research.

REFERENCES

[1] J. Bosch, "Software Architecture: The Next Step," presented at European Workshop on Soft. Arch. (EWSA), 2004.

[2] J. C. Dueñas and R. Capilla, "The Decision View of Software Architecture," presented at 2nd European Workshop on Software Architecture, Pisa, Italy, 2005.

[3] A. H. Dutoit, R. McCall, I. Mistrík, and B. Paech, "Rationale Management in Software Engineering: Concepts and Techniques," in *Rationale Management in Software Eng.*, A. H. Dutoit, R. McCall, I. Mistrík, and B. Paech, Eds.: Springer-Verlag Berlin Heidelberg, 2006, pp. 1-48.

[4] O. Gotel and A. Finkelstein, "Contribution Structures," *Proceedings of 2nd International Symposium on Requirements Engineering RE95*, pp. 100 - 107, 1995.

[5] A. Jansen and J. Bosch, "Software Architecture as a Set of Architectural Design Decisions," presented at Intl. Conference on Software Engineering (ICSE), 2005.

[6] P. Kruchten, *The Rational Unified Process: An Introduction*, 3 ed. Boston: Addison-Wesley, 2003.

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

[8] P. Kruchten, P. Lago, and H. Van Vliet, "Building up and reasoning about architectural knowledge," in *QoSA-Quality of Software Architecture*, vol. 4214, *LNCS*, C. Hofmeister, Ed. Vaesteras, Sweden: Springer-Verlag, 2006, pp. 43-58.

[9] J. Lee, "Design Rationale Systems: Understanding the Issues," *IEEE Expert*, vol. 12, pp. 78-85, 1997.

[10] I. Nonaka and H. Takeuchi, *The Knowledge Creating Company: How Japanese Companies Create the Dynamics of Innovation*. Oxford, UK: Oxford University Press, 1995.

[11] M.-A. Storey, L.-T. Cheng, R. I. Bull, and P. Rigby, "Waypointing and Social Tagging to Support Program Navigation," in *CHI '06 extended abstracts on Human factors in computing systems*. ACM Press, 2006, pp. 1367-1372.

[12] A. Tang, M. A. Babar, I. Gorton, and J. Han, "A Survey of Architecture Design Rationale," *Journal of Systems and Software*, vol. 79, pp. 1792-1804, 2006.