

Tool-Based Capture and Exploration of Software Architectural Design Decisions

by

Larix Lee

B.A.Sc., University of British Columbia, 2005

THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

The Faculty of Graduate Studies

(Electrical and Computer Engineering)

The University of British Columbia
(Vancouver)

February 2009

© Larix Lee, 2009

ABSTRACT

Developing software-intensive systems involves making many design decisions, some of which are decisions that govern the architecture of the system. Since changes to these architectural decisions affect many parts of the system being developed, design decisions pertaining to the system architecture should be documented and the knowledge the decisions contain should be explored. Many researchers and industry practitioners in the software architecture and maintenance communities have identified this need for design decision documentation as well as exploration. They have proposed design knowledge, rationale and decision representation models, suggested requirements, and determined uses and challenges to overcome when utilizing software architectural design decisions. Summarizing and integrating the various works of these researchers and industry practitioners would better represent the current state of research in exploring architectural knowledge and documenting design decisions, thereby creating a common foundation for new discoveries to be built. I present a new system-based tool that I developed called ADDEX, which attempts to unify the current discoveries, models, requirements, and guidelines for design decisions. In addition to integrating the various works together, the ADDEX tool is a system designed to take a holistic approach to decision capture and exploration by explicitly supporting customized decision capture processes for software development organizations. The tool also provides visualization support to promote a better understanding of the software architecture through several decision visualization aspects. I used ADDEX to acquire and display industry decision sets to demonstrate the ability of the tool-based solution to capture and explore software architectural design decisions. Combined with industry feedback, the decision sets help evaluate the tool and verify that ADDEX met the requirements and guidelines described by the various researchers and industry practitioners on which the integrated solution is based. Feedback from industry provides insight into decision capturing and the practical use of decision visualization.

TABLE OF CONTENTS

ABSTRACT	ii
LIST OF TABLES	v
LIST OF FIGURES	vi
LIST OF ABBREVIATIONS.....	vii
ACKNOWLEDGEMENTS.....	viii
DEDICATION.....	ix
CHAPTER 1 INTRODUCTION	1
1.1 SIGNIFICANCE.....	1
1.2 RESEARCH GOALS	2
1.3 CONTRIBUTIONS OF THIS THESIS.....	3
1.4 ORGANIZATION OF THIS THESIS	3
CHAPTER 2 KNOWLEDGE AND ARCHITECTURAL DESIGN DECISION REPRESENTATION.....	5
2.1 KNOWLEDGE AND DESIGN DECISIONS.....	5
2.2 DESIGN DECISION REPRESENTATION CHALLENGES AND REQUIREMENTS	7
2.3 DESIGN DECISION REPRESENTATION MODELS	10
2.3.1 Design Rationale.....	10
2.3.2 Design Decision Entities.....	11
2.4 COMPARING REPRESENTATION MODELS	12
2.5 SELECTING THE DECISION REPRESENTATION MODEL	16
2.5.1 Design Decision Ontology Model	19
CHAPTER 3 SYSTEM APPROACH TO DECISION CAPTURE AND EXPLORATION.....	22
3.1 CHALLENGES AND REQUIREMENTS FOR DESIGN DECISION SYSTEMS.....	23
3.1.1 Visualization Tool Requirements	27
3.2 USE CASES FOR DESIGN DECISIONS	29
3.2.1 Use Case Actors and Roles.....	29
3.2.2 Use Cases.....	31
3.3 SELECTING THE USE CASES	33
3.4 SELECTING THE SYSTEM REQUIREMENTS	35
3.5 MEETING SOME CHALLENGES	37
CHAPTER 4 DECISION CAPTURE AND VISUALIZATION SUPPORT	38
4.1 DECISION CAPTURE	39
4.1.1 Approaches to Decision Capture	40
4.1.2 Customized Decision Capture	47
4.2 DECISION VISUALIZATION	50
4.2.1 Visualization and Design Decisions	50
4.2.2 Essential Decision Visualization Aspects	52
4.2.3 Visualization and Use Cases.....	55

CHAPTER 5 ARCHITECTURAL DECISION TOOL DESIGN.....	58
5.1 TOOL DESIGN OVERVIEW.....	58
5.1.1 Decision Attributes	59
5.1.2 Users of the Tool	61
5.1.3 Decision Storage and Retrieval.....	61
5.2 DECISION CAPTURE TOOL IMPLEMENTATIONS	62
5.2.1 Formal Elicitation	63
5.2.2 Lightweight Top-down Capture.....	64
5.2.3 Lightweight Bottom-up Capture.....	67
5.3 DECISION VISUALIZATION TOOL IMPLEMENTATION.....	68
5.3.1 Decision / Relationship Lists	69
5.3.2 Decision Structure Visualization	70
5.3.3 Decision Chronology Visualization.....	72
5.3.4 Decision Impact Visualization.....	74
5.4 COMPARISON WITH OTHER CURRENT DECISION TOOLS.....	76
5.5 MEETING THE REQUIREMENTS	79
CHAPTER 6 EXPERIENCE WITH THE TOOLS.....	85
6.1 DEVELOPMENTAL SELF-TESTING	85
6.2 DECISION ACQUISITION.....	87
6.2.1 Industry Participants and Feedback	87
6.2.2 Decision Datasets and Findings.....	89
6.3 VISUALIZATION STUDY WITH INDUSTRY	91
6.3.1 Industry Participation.....	92
6.3.2 Feedback.....	93
6.4 TOOL USABILITY	93
6.4.1 Performing the Tasks.....	95
6.4.2 Observations and Analysis	98
6.4.3 Tool Refinements	101
6.5 DECISION CAPTURE TOOL COMPARISON EXPERIMENT.....	102
6.5.1 Experiment Overview	103
6.5.2 Experiment Results.....	103
CHAPTER 7 CONCLUSIONS AND SUMMARY	105
7.1 RESEARCH GOALS SUMMARY	105
7.2 CONTRIBUTIONS OF THIS WORK.....	106
7.3 FUTURE WORK	107
7.4 CONCLUSION	109
REFERENCES.....	111
APPENDIX A – ADDEX USER’S GUIDE.....	116
APPENDIX B – ETHICS APPROVAL	135
APPENDIX C – LIST OF PUBLICATIONS	138

LIST OF TABLES

TABLE 1: ARCHITECTURAL DESIGN DECISION REPRESENTATION CHALLENGES	8
TABLE 2: REQUIREMENTS FOR SOFTWARE DESIGN DECISION REPRESENTATION	9
TABLE 3: SUMMARY OF SEVERAL DESIGN REPRESENTATION MODELS	13
TABLE 4: MANDATORY AND OPTIONAL ATTRIBUTES OF ARCHITECTURAL KNOWLEDGE	14
TABLE 5: ESSENTIAL AND OPTIONAL DOCUMENTATION OF ARCHITECTURAL DECISIONS	15
TABLE 6: SUMMARY OF THE RESULTS FROM FALESSI'S TWO STUDIES ON DDRD INFORMATION IMPORTANCE.....	15
TABLE 7: GROUPING THE RESULTS OF FALESSI'S TWO STUDIES ON DDRD INFORMATION IMPORTANCE	16
TABLE 8: ATTRIBUTES OF DECISIONS	19
TABLE 9: DECISION RELATIONSHIPS	20
TABLE 10: DESIGN DECISION SYSTEM ISSUES AND CHALLENGES	24
TABLE 11: REQUIREMENTS FOR ARCHITECTURAL KNOWLEDGE AND DESIGN DECISION SYSTEMS	26
TABLE 12: VISUALIZATION TOOL REQUIREMENTS.....	27
TABLE 13: PASSIVE OR ACTIVE ROLES FOR ARCHITECTURAL KNOWLEDGE USE CASE ACTORS	30
TABLE 14: USE CASE ACTORS FOR ARCHITECTURAL KNOWLEDGE AND DESIGN DECISIONS.....	30
TABLE 15: ARCHITECTURAL KNOWLEDGE (DESIGN DECISION) USE CASES	32
TABLE 16: COMPARING KRUCHTEN'S LIST WITH VAN DER VEN'S LIST OF USE CASES.....	33
TABLE 17: SUMMARY OF SELECTED REQUIREMENTS AND USE CASES	36
TABLE 18: DECISION IMPACT MATRIX EXAMPLE	55
TABLE 19: USE CASES AND THE FOUR DECISION VISUALIZATION ASPECTS	56
TABLE 20: DESIGN DECISION ATTRIBUTES IMPLEMENTED IN EACH OF THE THREE CAPTURE TOOLS	60
TABLE 21: REQUIREMENTS TRACEABILITY MATRIX FOR ADDEX	80
TABLE 22: VISUALIZATION TOOL REQUIREMENTS MATRIX	83
TABLE 23: INDUSTRY PARTICIPANTS SUMMARY	88
TABLE 24: SEQUENCE OF ACTIONS PERFORMED FOR CERTAIN TASKS	96

LIST OF FIGURES

FIGURE 1: UML STATE DIAGRAM OF DECISION STATES AND THEIR TRANSITIONS	20
FIGURE 2: DECISION CAPTURE AND EXPLORATION RELATIONSHIP	22
FIGURE 3: GENERALIZATION OF FORMAL ELICITATION	41
FIGURE 4: A LIGHTWEIGHT TOP-DOWN CAPTURE METHOD.....	43
FIGURE 5: A LIGHTWEIGHT BOTTOM-UP CAPTURE METHOD	45
FIGURE 6: ADDEX SYSTEM DIAGRAM.....	59
FIGURE 7: UML DIAGRAM OF THE COMMON FRAMEWORK'S BASIC DECISION REPRESENTATION STRUCTURE	62
FIGURE 8: SCREENSHOT OF THE FORMAL ELICITATION TOOL	64
FIGURE 9: SCREENSHOT OF THE TOP-DOWN CAPTURE TOOL.....	65
FIGURE 10: SCREENSHOT OF THE BOTTOM-UP CAPTURE TOOL	68
FIGURE 11: DECISION AND RELATIONSHIP LISTS FOR A SET OF DECISIONS	70
FIGURE 12: DECISION STRUCTURE VIEW OF A SET OF DECISIONS.....	71
FIGURE 13: SEMANTIC ZOOMING IN THE DECISION STRUCTURE VIEW	72
FIGURE 14: CHRONOLOGICAL VIEW OF A SET OF DESIGN DECISIONS	73
FIGURE 15: CHRONOLOGICAL VIEW OF A SET OF DESIGN DECISIONS: CATEGORIZED BY AUTHOR	74
FIGURE 16: DECISION IMPACT VIEW OF DESIGN DECISIONS	75

LIST OF ABBREVIATIONS

ADD	Architectural Design Decision
ADDEX	Architectural Design Decision Exploration (tool)
ADDSS	Architecture Design Decision Support System
AK	Architectural Knowledge
DDRD	Design Decisions Rationale Documentation
DRCS	Design Rationale Capture System
DRL	Decision Representation Language
IBIS	Issue-Based Information Systems
ID	Identifier
IDE	Integrated Development Environment
gIBIS	graphical IBIS
InfoRAT	Inferencing Over Rationale
PHI	Procedural Hierarchy of Issues
QOC	Questions, Options, and Criteria
SEURAT	Software Engineering Using RATionale
SOA	Service-Oriented Architecture

ACKNOWLEDGEMENTS

To my supervisor Philippe, I express my deep gratitude for the opportunity to work with you as one of your graduate students. I thank you for your guidance, patience, and kind support during my graduate school experience and for the freedom to investigate what interests me. Many thanks go to those who generously funded my research (IBM, Ensemble Systems, NSERC, and ICICS), your support made this thesis possible. Sid, Kosta, thank you for reviewing my thesis and providing significant feedback on my research.

To all my friends I worked with in SEAL: David, Eve, Mandana, Steve, Yvonne, Davide, Agung, Jaana, Sam, and Erin, and the folks in the LERSSE side of the lab, I will always remember all those memorable times we had together. You've made the laboratory warm, supportive and "productive". Special thanks go to David for all the help and tips in my research, from computers to good eats, and to Mandana, who helped me sort through the research domain and braved reviewing my thesis. To Davide, thank you for inspiring and encouraging me through some difficult times. Your help in sorting out ideas and in empirical software engineering is greatly appreciated. To Patricia and Hans, thanks for the discussions and the opportunity to work with you. Rik: thanks for the enlightening talks and for the great memories in Amsterdam. Olaf: thanks for your helpful feedback and interesting ideas.

To Elaine and Teresa, I greatly thank you for all your help in the capture and visualization studies, from finding industry contacts to helping me sort through some painful datasets. I especially would like to thank the four study participants and all those who were involved: CW, DM, FP, JS, NH, PVA and SM, this research would not have been completed without all of you. Your time and contribution is greatly appreciated.

To Luis, I enjoyed being a part of your operating systems class both as an undergraduate student then and as an assistant the past few years. Your advice has always been helpful. To Lee, thanks for the mind-opening discussions over coffee, in and out of class.

To Johnson, thank you for being a great friend, and for all the munchies. Nelson, thanks for the reminders to have fun. Wilson, Pamela, and Joyce, thanks for being supportive.

But the greatest thanks of all go to God and then to my family who always prayed for me and supported me: Mom, Dad, Maple (who also willingly endured proofreading all my writing more than once), and my late grandma (Poh-Poh).

Dedicated to God and to my family –

Mom, Dad, Maple, and Grandma (Poh-Poh)

CHAPTER 1

INTRODUCTION

Designing is a process of making decisions; decisions build on each other and result in a final design. However, the dynamic nature of software development means that software designs are often never “final” but continue to change and grow, causing old decisions to become obsolete as new decisions are made. As a result, software developers need to cope with requirement and architecture changes, design evolution, and the consequences on implementation. Changing design decisions pertaining to the architecture of a software-intensive system may significantly affect the entire system being developed because architectural design decisions crosscut many aspects of the system or because they affect the foundations on which the system is built. Therefore, the need to capture and manage software architecture design knowledge is important in any software development organization, and it is even more important in large organizations with high personnel turnover where key staff members, such as architects and other designers, have moved on to other projects and took their design knowledge with them.

1.1 Significance

Capturing architectural design knowledge could ease the burden of understanding the design as the original designers had envisioned. Moreover, recent works in software architecture research is increasingly recognizing the role of design decisions to represent software architecture. With support from the software architecture community and the push for better knowledge documentation, the research focus is shifting to the capturing and use of software architectural design decisions. The architectural knowledge provided by these design decisions is useful throughout the entire development organization to include reviewers, programmers, testers, maintainers, and support. Exploring and using captured decisions to find additional information hidden within the decisions are a fundamental goal of research in this area.

Although there is an increasing amount of research in representing, capturing, and managing design decisions in software design and development, many works address and target specific aspects of architectural design decisions, resulting in limited exploration and assessment of effectiveness for those contributions. I define architectural design decision exploration to be the group of activities performed on captured decisions that include discovery (by other people), perusal, understanding, and learning of architectural design decisions and the architecture they represent. Architectural design decision exploitation (the analysis, extrapolation and creation of new information and design decisions from captured decisions) is included in this group of activities. However, both decision capture and exploration depend on each other, as it is difficult to capture decisions without knowing how they are used and explored, yet the usefulness of exploring design decisions depend on having them captured beforehand. Therefore, a holistic approach is needed to investigate software architectural design decisions. By bringing together the current works and contributions of researchers and industry practitioners in the area of design decision capture and exploration, we can better assist software development organizations manage and design software systems.

1.2 Research Goals

The objective of this thesis is to integrate several research contributions and works in the field involving software architectural design decisions to come up with an integrated tool-based solution for software architectural design decision capture and exploration. The solution addresses as much of the requirements, recommendations, and guidelines recently proposed by various members of the research community. I intend to achieve this objective by:

- Determining a common decision representation model for decision capture and exploration
- Identifying common challenges and issues among the contributions in the scope of software architectural design decisions and architectural knowledge
- Determining common requirements and use cases of architectural design decisions
- Implementing the above set of requirements and use cases as a tool-based solution to demonstrate the integrated set of requirements and use cases
- Evaluating the implemented solution with industry datasets and industry practitioners

1.3 Contributions of This Thesis

The contributions of this thesis are:

- A solution that integrates the current and common issues, challenges, requirements, use cases, and guidelines to capture and explore design decisions in a system-based approach. This solution represents the most current view of design decision systems in the field of software architecture and maintenance
- A proposal of using three capture approaches (together or separately) to encourage and facilitate decision capture:
 1. Formal elicitation,
 2. lightweight top-down, and
 3. lightweight bottom-up
- A proposal of four visualization aspects that apply to software architectural design decisions to promote decision exploration:
 1. Tabular lists,
 2. decision structure visualization,
 3. decision chronology visualization, and
 4. decision impact visualization
- A tool called ADDEX that implements the integrated solution in the context of a tool that supports the capture and exploration of architectural design decisions:
 - Provides an integrated environment for decision capture and exploration
 - Supports decision capture across various stages of the development process
 - Visualizes four aspects of design decisions to support decision exploration
- A demonstration of the implemented tool to capture and represent industry datasets
- An evaluation of the tool by industry practitioners to gain feedback on the tool-based solution and the proposed decision capture approaches and decision visualization aspects

1.4 Organization of This Thesis

The following six chapters of this thesis describe how we can capture and explore software architectural design decisions using the ADDEX tool. Starting with Chapter 2, I discuss what architectural knowledge is and how we can represent architectural knowledge as design

decisions. This chapter also describes the selection of the decision representation model. Chapter 3 focuses on approaching decision capture and exploration from a system perspective, while highlighting the challenges, requirements, and use cases that researchers have recommended to follow for design decision systems. Also described in this chapter is how I selected the use cases and system requirements to meet some of the described challenges. In Chapter 4, I propose additional guidelines and context for decision capture and decision visualization, then I describe the ADDEX tool that I implemented in Chapter 5 to best fulfill the chosen requirements and use cases. Chapter 6 looks at the initial practical experience with the tool and describes a simple evaluation of the tool with industry practitioners. Chapter 7 summarizes and concludes my research in the tool support for the capture and exploration of software architectural design decisions.

CHAPTER 2

KNOWLEDGE AND ARCHITECTURAL DESIGN

DECISION REPRESENTATION

This thesis defines software architectural knowledge to be the knowledge pertaining to the software architectural design as well as the set of design decisions that resulted in that architectural design (Kruchten *et al.*, 2005). Architectural design decisions are decisions that cross-cut multiple components and connectors, and intertwine with other design decisions (Jansen & Bosch, 2005), such that changing one architectural decision could affect other decisions. In other words, architectural design decisions are design decisions that pertain to the overarching goals and characteristics of the system. If we seek to help people understand a software design through architectural design decisions, then we require an understanding of knowledge and how we can represent architectural knowledge as design decisions.

2.1 Knowledge and Design Decisions

Knowledge itself is difficult to define: great philosophers from Plato to Polanyi have wrestled with the definition of knowledge. Although there are many definitions of knowledge, the ones that focus on the process of knowing would help explain the difficulties of capturing architectural knowledge. (Polanyi, 1966) defines two forms of knowing (awareness) in his book, “The Tacit Dimension”: tacit and focal (Grant, 2007). In this view, knowledge brought to the focus of attention is defined as focal knowledge. Focal knowledge is easily expressed, shared and made apparent. Tacit knowledge, on the other hand, is difficult to express, so it cannot be easily communicated. Nonaka later contributed a similar definition, but expresses it as tacit and explicit knowledge. According to this definition, tacit knowledge is “highly personal... and difficult to communicate to others”, while explicit knowledge is “formal and systematic... [and] can be shared” (Nonaka, 1991). Nonaka expresses the interactions of the two forms of knowledge as processes of converting from one form to another (that is, tacit to tacit, tacit to explicit, explicit to tacit, and explicit to explicit) and illustrates the idea by applying the

processes to how corporate companies generate knowledge. Nonaka states that articulation (implicit to explicit) is vital for knowledge creation and communication in an organization.

Another perspective of Polanyi and Nonaka's definitions of knowledge is more concrete and it categorizes knowledge into three levels: tacit, documented, and formal (Kruchten *et al.*, 2006). Documented knowledge is knowledge that is captured in some form *outside the minds of people*. For example, documented knowledge can be the unstructured content found in a diary or notebook. Formalized knowledge is a particular case where the knowledge is documented and *structured in an organized, systematic fashion*, like a dictionary or an event logbook, so that finding patterns or making associations within the data can exploit it. The third level of knowledge, tacit knowledge, is acquired from experience and is difficult to express. Tacit knowledge *remains in the mind*, where it can be forgotten. Knowledge that pertains to preferences and choices are often not documented and hence remains tacit.

We can apply knowledge classification to software development. Basically, software is a product of sequenced operations and declarations. Data structures are used to organize the operations and declarations, while sets of operations and declarations can be logically evaluated against each other. Furthermore, those operations and declarations can be grouped together into files, classes, and packages, and the careful selection of which groups of operations and declarations result in design patterns and architecture. (Robillard, 1999) defines five knowledge concepts that are applicable to software knowledge: procedural/declarative, schema, proposition, chunking, and planning. The procedural/declarative concept is the content, or essence, of the knowledge. Procedural refers to sequences of actions and events, and declarative describe a meaning or experience. The schema concept abstracts the first by organizing and classifying knowledge by similarity. Abstracting further, the concept of propositions is used to represent knowledge formally where information could be affirmed, while procedures, declarations and propositions can be collected, grouped or sub-grouped together to limit scope for easier understanding. The highest level of abstraction is the planning concept, where plans help manage knowledge by defining goals and determining which groups of knowledge are needed to achieve the goals. In essence, software planning is making architectural design decisions.

The software maintenance community has researched into the issues relating to design erosion and lost knowledge in software design for many years (van Gurp & Bosch, 2002). The research interested members of the software architecture community, which sparked further research in architectural knowledge. A recent literature survey by (de Boer & Farenhorst) collected and synthesized definitions of architectural knowledge to conclude that a significant part of the knowledge involves the use of design decisions.

2.2 Design Decision Representation Challenges and Requirements

Unfortunately, representing software architectural knowledge using design decisions is not an easy task; a number of researchers identified some decision representation challenges for use with software architecture. These challenges include encountered issues, concerns, and common themes that should be addressed by the research community. Table 1 below highlights some of these challenges that are found in current literature. One challenge is the lack of a first-class representation (Bosch, 2004) for design decisions within software architecture, where we can refer to and manipulate design decisions as unique, fundamental entities. The structure of the first-class representation allows the design decision, its rationale and its assumptions to be accessed, analyzed, generalized, and contextualized more readily simply by making the decisions explicit. A significant challenge is how to deal with the dynamic nature of design: the decision representation must be able to support and keep track of design changes with minimal decision management overhead. The decision representation must handle design changes while maintaining clarity and simplicity to convey the changes plus the resulting implications and consequences to the affected stakeholders. However, addressing this challenge involves more than just creating a satisfactory representation model, it also depends on how people would create and utilize the design decisions. The designers' participation, the organizations' needs and development processes contribute to how much decisions are captured and how well the decisions are represented. (Tang *et al.*) recommend that future work in decision representation should involve generalizing design rationale into types, investigating decision representation methodologies and tools, as well as assessing needs for decision documentation.

Table 1: Architectural design decision representation challenges

Topic (Source)	Challenges
Design knowledge representation challenges (Regli <i>et al.</i> , 2000)	<ul style="list-style-type: none"> ▪ Finding the best method to assist designers to make decisions ▪ Representing design knowledge as system components ▪ Representing features by context or group ▪ Generalizing rationales with generic clauses ▪ Using a formal representation language ▪ Supporting both decision authoring and browsing ▪ Personalizing the captured decisions
Decision representation challenges (Bosch, 2004)	<ul style="list-style-type: none"> ▪ Lack of first-class representation ▪ Design decisions are cross-cutting and intertwined ▪ High cost of change ▪ Design rules and constraints violated ▪ Obsolete design decisions not removed
Architecture decisions issues (Tyree & Ackerman, 2005)	<ul style="list-style-type: none"> ▪ Conveying change ▪ Conveying implications ▪ Conveying rationale & options ▪ Ease of traceability ▪ Providing agile documentation
Areas for future investigation in architecture design rationale (Tang <i>et al.</i> , 2006)	<ul style="list-style-type: none"> ▪ Different types of design rationale ▪ Designer's attitude ▪ Necessity for design rationale documentation ▪ Design rationale methodology support ▪ Design rationale tool support

There are recent studies that discuss how the information should be represented as a software architectural design decision. Table 2 shows several sets of requirements that researchers have proposed to represent software design decisions. (J. Lee, 1997) describes three layers that make up the generic structure of design rationale representation: decision layer, design artifact layer, and design intent layer. Together, the three layers would also help document the functional dependencies of the design. Representation formality also plays a large role in the selection of what type of information to capture. The lower the formality, the easier it is for a person to express his or her design rationale; however, it becomes more difficult for a computer system to parse the data.

(Regli *et al.*) refer to three qualities for knowledge representation (ease of input, effective view, and activeness (Conklin & Burgess-Yakemovic, 1996)) that deal with the usability of design rationale and suggests that a formal design knowledge language be used while supporting design feature and rationale generalization. For architectural design decisions, (Bosch) states that the restructuring effect, design rules, design constraints, and rationale make up the four relevant aspects of design decisions. (Kruchten, 2004) views decisions as a set, describes decisions to

have a temporal flow (via change histories) and suggests the explicit recognition of decision relationships as a fundamental component of decisions. More recently, the architectural knowledge community worked out a succinct set of essential and optional information to document architectural design decisions (Avgeriou *et al.*, 2007). The essential components for design decisions are the decision description, the issue, the rationale, and the discarded options. Other information types like relationships, categories, or versioning are optional, but beneficial if captured.

Table 2: Requirements for software design decision representation

Topic (Source)	Requirements
Generic structure of explicitly representing design rationale (J. Lee, 1997)	<ul style="list-style-type: none"> ▪ Decision layer (argumentation, alternative, and evaluation) ▪ Design artifact layer ▪ Design intent layer
How to represent rationales (J. Lee, 1997)	<ul style="list-style-type: none"> ▪ Informal (captures unstructured, natural, raw form) ▪ Semi-formal (only parts are computer readable) ▪ Formal (All info rationale system can read and use)
Knowledge representation (Regli <i>et al.</i> , 2000)	<ul style="list-style-type: none"> ▪ Three qualities of representation: (Conklin & Burgess-Yakemovic, 1996): <ul style="list-style-type: none"> ▪ Ease of input ▪ Effective view ▪ Activeness (automatic action in response to events or conditions) ▪ Should have capability to represent potentially relevant features and combine features of objects in specific concepts to form coherent explanations ▪ Encode the modeling language in a form that can be shared with other applications and systems ▪ Formal language must be developed ▪ Systems should provide different views
Four relevant aspects of design decisions (Bosch, 2004)	<ul style="list-style-type: none"> ▪ Restructuring effect ▪ Design rules ▪ Design constraints ▪ Rationale
Ontology of design decisions (Kruchten, 2004)	<ul style="list-style-type: none"> ▪ Decision classes (existence/ban, property, executive) ▪ Decision attributes (Epitome, rationale, scope, state, history, cost, risk) ▪ Decision inter-relationships (see Table 9)
Conceptual model of a design decision (Avgeriou <i>et al.</i> , 2007)	<ul style="list-style-type: none"> ▪ A concern (can be broken into issues) ▪ The issue and its option(s) ▪ The decision (and inter-decision dependencies) ▪ Rationale(s) ▪ Option(s)

2.3 Design Decision Representation Models

As the researchers develop requirements and guidelines to represent design decisions, they propose new representation models or build on other models that support their ideas. In the software community, two categories roughly divide the list of decision representation models: design rationale and decision entities. What distinguishes the two categories apart is the models' focus. The former category focuses on the background and context of a design decision while the latter category focuses on the decision itself. Representing decisions as entities is a more recent research progression in the software architecture community, whereas design rationale has roots to the software maintenance community. The subsections below provide a brief summary of the different types and the research progression of the decision representation models.

2.3.1 Design Rationale

Design rationale is “the historical record of the analysis that led to the choice of the particular artifact or the feature in question” (J. Lee & Lai, 1996). Design rationale can explain the behaviour of a component, or the rationale can refer to non-functional requirements and imposed system constraints, such as response-time or interoperability. Many research works in design rationale recommend the use of an argumentation structure, which improves the capturing process as the knowledge can be expressed in familiar forms. Pioneered by the earlier works on argumentation and decision making processes by (Kunz & Rittel) with Issue-Based Information Systems (IBIS), (which use structured elements such as issues, positions, arguments,) many works relating to capturing knowledge in software development processes and maintenance emerged by the mid 1980’s. (Potts & Bruns) adopt the IBIS model in their issue-based model of design deliberation that is investigated and extended by many other methods and models, such as the Procedural Hierarchy of Issues (PHI) approach, as referenced by (Fischer *et al.*, 1989). PHI is essentially a recursive definition of the IBIS model, which takes into consideration subsets of issues and solutions found in problem domains. The concept of decision structures and dependency networks in a support environment for software maintenance are also investigated to assist software engineers in understanding the design and the choices made (Wild & Maly, 1988, Wild *et al.*, 1989). (Conklin & Begeman) implement a hypertext tool known as graphical IBIS (gIBIS) that utilizes the IBIS method to explore the capture of design rationale, supplementing IBIS slightly to focus more on the decisions made during design. The gIBIS tool allows

computer-supported collaboration and investigates how to navigate large sets of rationale. (Fischer *et al.*) create a hypertext tool that uses the capabilities of the PHI approach and hypertext to design deliberation.

However, IBIS and its derivatives do not satisfy all members of the design community. (MacLean *et al.*) find that the IBIS-based approaches do not fully apply to design spaces, so they propose the Questions, Options, and Criteria (QOC) approach to address those needs. The QOC approach uses more structured elements to describe design rationale, where the QOC approach includes questions, options, criteria, assessments, arguments, and decisions (Dutoit *et al.*, 2006). Other approaches to design rationale capture were investigated. A concept known as decision rationale was introduced by (J. Lee, 1990), where the work surrounds the concept that decision rationale is a subset of design rationale. (J. Lee, 1990) describes a way to represent the decisions through the Decision Representation Language (DRL) and he demonstrates it using the SIBYL tool (J. Lee, 1991). A few years later, (Klein) introduces the Design Rationale Capture System (DRCS) model which, like DRL, focuses more on the decisions than on the issues.

In the early 2000's, research into design rationale focus more on the capture and manipulation of design rationale. The InfoRAT (Inferencing over Rationale) tool (Burge & Brown, 2000, Burge & Brown, 2001), and the RATSpeak rationale representation language in the SEURAT (Software Engineering Using RATionale) tool (Burge & Brown, 2004) focuses on decision design rationale during implementation and maintenance phases. (Dutoit & Paech) describes the use of design rationale using the QOC approach during the specification of use cases. The Sysiphus tool by (Wolf & Dutoit) investigates the capture of design rationale throughout a software organization. The results of empirical investigations demonstrate that design rationale documentation is useful (Karsenty, 1996), improves change-task completion rates and quality (Bratthall *et al.*, 2000), and is efficient and effective (Falessi *et al.*, 2006).

2.3.2 Design Decision Entities

The shift from issue-based to decision-based representation of architectural knowledge and software design is demonstrated through the development of the DRL and DRCS models. Soon after, a new research direction emerges and the software architecture and maintenance

communities begin switching from capturing issue-based design reasoning towards more formalized capture of design decisions. Representing design decisions explicitly focuses on the choices as a primary objective, while both the context and the justification are secondary to that decision. Although research in the design rationale community deals with representing decisions and assumptions explicitly, the software architecture community develop this area significantly due to the software architectural shift towards making design decisions and assumptions explicit. An approach to making decisions first-class entities is described by (Bosch) followed by Kruchten with his design decision ontology (Kruchten, 2004). (Tyree & Ackerman) define an architecture decision description template that describes a set of attributes used to represent a decision. These attributes include the issues, decisions, statuses, assumptions, constraints, positions, arguments, implications, and the related decisions, requirements, artifacts, and principles. The Archium metamodel (Jansen & Bosch, 2005) focuses on architectural changes by linking software architectural components, requirements, and decision models together using explicit change deltas. A metamodel proposed by (Lago & van Vliet) integrates the idea of assumptions with design decisions and focuses on capturing cross-cutting concerns by modelling invariabilities made during design. The metamodel in ADDSS (Architecture Design Decision Support System) (Capilla *et al.*, 2006) focuses on the relationships between decisions, architecture, stakeholders, and requirements. The architecture ontology of (Akerman & Tyree) applies the decision model to architectural assets by linking stakeholder concerns, assumptions, alternatives, and assets together.

2.4 Comparing Representation Models

As there are various design decision representation models available, determining which decision model to use for decision representation can be difficult. Rationale-based decision representation can be used when the focus of the decision documentation is on design reasoning, whereas the Archium metamodel is better suited to describe the progression of design decisions through changes in architectural components and requirements. Since there are many decision representation models, it is difficult to see which model captures what types of information. Table 3 highlights the key attributes of several design decision representation models.

Table 3: Summary of several decision representation models

Source	Main attributes of the representation model	
IBIS-based (Kunz & Rittel, 1970)	<ul style="list-style-type: none"> ▪ Issues ▪ Positions 	<ul style="list-style-type: none"> ▪ Arguments
Potts and Bruns model (Potts & Bruns, 1988)	<ul style="list-style-type: none"> ▪ Artifact ▪ Issue 	<ul style="list-style-type: none"> ▪ Alternative ▪ Justification
DRL (J. Lee, 1990)	<ul style="list-style-type: none"> ▪ Artifact ▪ Alternative ▪ Goal ▪ Issue ▪ Claim 	<ul style="list-style-type: none"> ▪ Question ▪ Group ▪ Procedure ▪ Viewpoint
QOC (MacLean <i>et al.</i> , 1991)	<ul style="list-style-type: none"> ▪ Questions ▪ Options ▪ Criteria 	<ul style="list-style-type: none"> ▪ Assessments ▪ Arguments ▪ Decisions
Ontology of design decisions (Kruchten, 2004)	<ul style="list-style-type: none"> ▪ Decision classes (Existence/ban, property, executive) 	<ul style="list-style-type: none"> ▪ Decision attributes (Epitome, rationale, scope, state, history, cost, risk) ▪ Decision interrelationships (see Table 9)
Architectural design decision model in Archium (Jansen & Bosch, 2005)	<ul style="list-style-type: none"> ▪ Problem ▪ Motivation ▪ Cause ▪ Context ▪ Decision 	<ul style="list-style-type: none"> ▪ Architectural modification ▪ Potential solutions (Description, design rules, design constraints, consequences, pros/cons)
Architecture decision description template (Tyree & Ackerman, 2005)	<ul style="list-style-type: none"> ▪ Issue ▪ Decision ▪ Status ▪ Group ▪ Assumptions ▪ Constraints ▪ Positions 	<ul style="list-style-type: none"> ▪ Arguments ▪ Implications ▪ Related decisions ▪ Related requirements ▪ Related artifacts ▪ Related principles ▪ Notes
Generic design rationales (Tang <i>et al.</i> , 2006)	<ul style="list-style-type: none"> ▪ Design constraints ▪ Design assumptions ▪ Weakness (of a design) ▪ Benefit (of a design) ▪ Cost (of a design) 	<ul style="list-style-type: none"> ▪ Complexity (of a design) ▪ Certainty of design ▪ Certainty of implementation ▪ Tradeoffs

The general theme for design rationale representation models, with acknowledgement of some vocabulary differences between the models, is on design deliberation, showing issues, options, goals, and assessments. For example, the QOC “decision” is simply the selection of an alternative that answers an issue. Likewise, for the decision entity representation, the majority of the decision attributes can be represented entirely or as a part of another attribute, such as “epitome” and “decision”, or decision “status” as a subset of “design certainty”. Both the design rationale and decision entity representation models address the concept of a choice to be made, followed by some justification or rationale behind that choice. However, some models have

attributes that others do not, such as the explicit “criteria” attribute in QOC , or the “architectural modification” in Archium, as well as the “decision interrelationships” in the decision ontology model. These differences help a representation model address specific situations, needs, and emphases of a particular software development organization.

Different software development organizations would place different emphases on the type of information captured to document their design decisions; what is optional to one organization is mandatory to another. (Capilla *et al.*, 2007) address this issue directly by proposing a flexible approach to what constitutes architectural knowledge to suit the needs of the organization. This flexible approach attempts to integrate the various works in documenting software architectural knowledge and describes twenty-five mandatory and optional architectural knowledge attributes. Since the definition that architectural knowledge includes a set of decisions, most of these attributes apply to architectural design decisions as well. The results are summarized in Table 4. According to Capilla *et al.*, eight attributes should be documented in the captured design decision and seventeen attributes can be documented depending on the documentation needs of the organization. Five of these optional attributes involve the evolution of a design, by documenting the chronology, versioning, validity, ratings, and traceability of the knowledge.

Table 4: Mandatory and optional attributes of architectural knowledge. Capilla, Nava, and Dueñas determined eight attributes that should be defined in an architectural design decision at all times during the life of the system, and lists seventeen attributes that they classify as optional where five of these attributes are useful during design evolution.

Mandatory	Optional
<ul style="list-style-type: none"> ▪ Decision name/description ▪ Constraints ▪ Dependencies ▪ Status ▪ Rationale ▪ Design patterns ▪ Architectural solution ▪ Requirements 	<ul style="list-style-type: none"> ▪ Alternative decisions ▪ Assumptions ▪ Pros / cons ▪ Category of decisions ▪ Iteration ▪ Project/software architecture information
Design evolution	
	<ul style="list-style-type: none"> ▪ Date/version ▪ Obsolete decision ▪ Validity
	<ul style="list-style-type: none"> ▪ Responsible ▪ Architecture view ▪ Stakeholders ▪ Related principles ▪ Notes ▪ Quality attributes

At the same time, the architectural knowledge community worked together to describe a rough set of essential and optional documentation information (shown in Table 5) for architectural design decision representation (Avgeriou *et al.*, 2007).

Table 5: Essential and optional documentation of architectural decisions (Avgeriou *et al.*, 2007)

Core (Essential)	Relationships	Management
<ul style="list-style-type: none"> ▪ Decision description ▪ Issue ▪ Rationale ▪ Discarded options 	<ul style="list-style-type: none"> ▪ Links and relationship types to other decisions ▪ Traceability to requirements, design, implementation, and tests ▪ Categories 	<ul style="list-style-type: none"> ▪ Name, ID, system, author, owner, etc. ▪ Version history ▪ Status ▪ Decision type ▪ Result cost or risk analysis

A recent study performed by (Falessi *et al.*, 2008a) identifies what a group of software developers (graduate students) determine most important to capture in the design decisions rationale documentation (DDRD) information. The DDRD information uses most of the information categories listed in the architecture decision description template (Tyree & Ackerman, 2005). The experiment was later replicated as part of a follow-up study (Falessi *et al.*, 2008b). Comparing these two studies provides an idea of what categories of information is generally found to be useful. Tables 6 and 7 below summarize the differences.

Table 6: Summary of the results from Falessi's two studies on DDRD information importance (Falessi *et al.*, 2008a, Falessi *et al.*, 2008b). The feasibility study is performed first and is later replicated in another study with another set of study participants who would better represent software professionals.

DDRD information	Feasibility study		Replicated Study	
	Mean (%)	Ranking	Mean (%)	Ranking
Issue	71	2	91	1
Decision	94	1	79	2
Status	47	7	25	9
Assumptions	43	8	49	7
Constraints	22	10	54	6
Positions	54	6	72	4
Argument	65	4	67	5
Implications	38	9	21	= 10
Related decisions	56	5	28	8
Related requirements	68	3	74	3
Related artifacts	9	12	14	12
Related principles	12	11	21	= 10
Notes	0.5	13	5	13

Table 7: Grouping the results of Falessi's two studies on DDRD information importance. The category importance groupings were created by finding the group boundaries that result in the minimum number of category ranking changes for both study results.

General groups of importance	Avg. # of rank changes $\frac{\sum^n_i (Rank_{rep1} - Rank_{feas1})}{N}$ (Lower # is more confident)	DDRD information (**) (n = num of categories in each group)	
High importance	$(1+1)/2 = 1.0$	- Design Issues	- Design decisions
Medium-high importance	$(0+1+2+3)/4 = 1.5$	- Related requirements - Positions	- Arguments - Related decisions
Medium low importance	$(2+1+1+3)/4 = 1.75 (*)$	- Status - Assumption	- Implications - Constraint
Low importance	$(0+0+0)/3 = 0.0 (*)$	- Principles - Artifacts	- Notes

* - To simplify the complexity caused by a two-way tie in the category rankings of the replicated study, the lowest difference for the affected categories is used.
**- The selection of which group each DDRD Information category belongs to is determined by finding the boundaries that minimizes the amount of grouping changes between the two separate studies.

There are differences between Capilla's list (Table 4), Avgeriou's list (Table 5) and the grouping of Falessi's results (Table 7), but the main differences can be attributed to the use of vocabulary. For example, Falessi's “issues”, “arguments”, and “positions” and Avgeriou's “discarded options” can be addressed by Capilla's “rationale”. Likewise, the “constraints” and “related decisions/requirements” are addressed by “dependencies”. With the acknowledgement of these differences in mind, the general consensus is that the decisions, rationale, status, requirements, and dependencies are considered important to capture, while the remaining attributes are considered unimportant (like “notes” and “artifacts”) or vary depending on the individuals, as demonstrated by Falessi's findings. Tailoring the amount of knowledge to capture based on the values of an organization (Falessi *et al.*, 2008b) may address the various needs and uses for the captured architectural knowledge.

2.5 Selecting the Decision Representation Model

As there is no one right approach to represent design decisions (Regli *et al.*, 2000), I need to choose a decision representation model that can best service the scope of my thesis. Since my research is in the area of architectural decision capture and exploration, the selection of the right

representation model guides the decision capture process and establishes a high decision exploration potential. My decision is to find an existing decision representation model to leverage the predefined/peer-reviewed as a cost-saving and risk reducing measure, yet I also acknowledge that there are limitations to each model. The model selection required much careful thought when I began my research.

When the work of this thesis started, there were only a few decision models that I deemed sufficiently detailed for use. These choices were IBIS, PHI, QOC, DRL, or DRCS from the design rationale stream, while from the decision entities stream there were only Kruchten's decision ontology model, Tyree and Ackerman's decision description template, the Archium metamodel, and Lago and van Vliet's "assumptions" metamodel. Using decision entities to represent decisions provide a guiding structure to facilitate decision capture while providing visualization, manipulation, and temporal support to understand and manage design decisions. However, only DRL, DRCS, the decision ontology model, and the decision description template address design decisions explicitly. For design rationale, design decisions are embedded and can be lost in the justification texts.

I exclude rationale-based decision models since those models detract attention from the core decisions. For a study in decision exploration and analysis, a simpler, broader decision model is preferred so that software architects and designers can document various types of information during various stages of software development. This excludes the Archium metamodel, despite its explicit support for architectural changes, because it is closely linked to the architecture model. I also exclude models where decisions cannot be described in greater detail. I adopt Kruchten's decision model for my research work because the model is simpler and the decisions can be presented separately from the architectural context. The model's decision states and change logs make the model process-focused, and it is the only model at that time that explicitly represents decision relationships. Decision relationships could increase the exploratory and analytical potential of design decisions by providing additional associations and traceability, allowing a more rich foundation for visualizing relationships.

Research into design decision representation brings in new suggestions and decision models. The ADDSS metamodel is introduced by (Capilla *et al.*, 2006) and it attempts to unify the various design decision models with a flexible definition of the mandatory and optional characteristics of an architectural design decision (Capilla *et al.*, 2007). This definition gives much freedom in defining what type of information we need to capture and represent a design decision. Only one mandatory decision attribute (“design patterns”) is not explicitly represented in Kruchten’s decision model; rather, design patterns can be implicitly represented by modelling the design pattern as a decision itself (e.g., “Use the strategy pattern for all data model interfaces”). Research in the field of service-oriented architecture (SOA) includes applying design decision models as a part of SOA. For example, researchers from IBM suggest design decisions models to be a mean for SOA analysis and design (O. Zimmermann *et al.*, 2007). IBM Research also jointly investigates architectural decision modelling through the development of the AD_{kwik} tool, which is the subject of a doctoral thesis for (Schuster). The AD_{kwik} tool uses a decision model to model decisions, alternatives, and outcomes. Moreover, this model describes the relationships of decisions with their alternatives and outcomes into three dependency types—topic, time, and outcome. The latter two dependencies can also describe the influences of decisions on other decisions as well. The decision model for AD_{kwik} and Kruchten’s decision model are currently the only software architectural design decision models that explicitly represent inter-decision relationships.

During the course of my research in decision capture, I identify a couple more aspects that a design decision representation model should support: decision confidentiality and explicit support for incomplete decision documentation (L. Lee & Kruchten, 2007, L. Lee & Kruchten, 2008a). I suggest that the decision model should support different levels of disclosure, such as “personal”, “organization-wide” or “public”, where the selective-release of the decisions allow the gradual capture and formation of design decisions while reducing the effects of documenting personal or politically-charged decisions. Support for personal decisions benefits the capturer by providing an environment where the capturer can feel safe documenting their decisions. Moreover, the decision model should have the capability to explicitly keep decisions as tacit as possible, documenting only the essentials (like the name of a knowledgeable person) so that other people could find out who could answer their questions about a particular area. This

capability increases convenience in some situations where it is more time-efficient when the captured decision is conceptually difficult and can be better explained in person. Unfortunately, I identified these other aspects when my research was well underway so I was unable to integrate decision disclosure levels into the decision representation model I selected. However, with my proposal of customized decision capturing processes (see Section 4.1) I was able to include explicit support for incomplete decisions.

2.5.1 Design Decision Ontology Model

A brief explanation of the decision representation model used in my research work is warranted. The decision representation model makes no distinction between decision types — there is no concept of decision classes or hierarchy among decision entities. Each architectural design decision entity has attributes to describe the decision. These attributes and how they are represented are summarized in Table 8.

Table 8: Attributes of decisions

Name	Type
Epitome	Text
Rationale	Text or pointer
Scope	Text
State	Enumeration
History	List of (time stamp + author + change)
Categories	List
Publicity Level	Enumeration
Source (or expert)	Text

The epitome describes the essence of the decision and is supported by reasons stated in the rationale; however, the decision context is restricted by the scope of the decision. Each decision has a certain state, which describes the “maturity” of the decision. The states and its transition paths are depicted in Figure 1. Any change made to the decision attributes are logged in the decision history. The category attribute complements the decisions with additional information. The publicity level attribute sets the level of decision disclosure for the selective-release of design decisions, while the source/expert attribute can document where the knowledge is found for traceability or to support decisions intentionally left tacit.

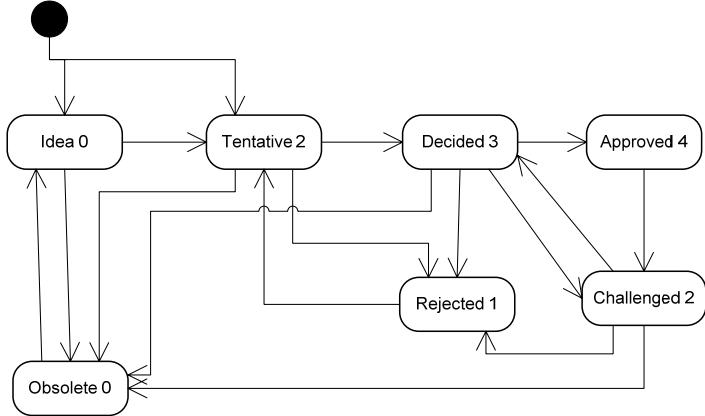


Figure 1: UML state diagram of decision states and their transitions. The number next to each state name is the promotion level for each state. Higher numbers mean greater levels, implying a higher decision “weight”. Arrows leading out from a state denote the transition paths for that decision state. Created decisions start out in the “idea” or “tentative” states. Decisions are never removed; they are given a new state (“rejected” or “obsolete”). Figure from (Kruchten, 2004), with permission.

However, there is one major aspect of the decision representation model that many other works fail to pick up on—decision relationships. In my model, there are ten inter-decision relationships. Table 9 shows the ten relationship classifications between decisions, and these relationships are of the form, “Decision A ‘is related to’ Decision B”.

Table 9: Decision relationships

Relationship Type	Association
Constrains	Directional
Forbids	Directional
Enables	Weak directional
Subsumes	Directional
Conflicts with	Bidirectional
Overrides	Directional
Comprises (is made of)	Directional
Is bound to	Strong bidirectional
Is an alternative to	Directional
Is related to	Weak directional

Decisions can constrain one another, where the affected decision is contingent upon the constraining decision. The weak form of this relationship is known as the enabling relationship,

while the bi-directional form is strong and is known as the binding relationship. Decisions could also forbid another decision from being made, or could be subsuming in that it can be more encompassing than another. Decision conflicts are symmetrical and are possible when both decisions are mutually exclusive and have the same scope. Although similar in description, alternatives differ from conflict relationships. Alternatives are decisions that address the same issue and scope, but can be replaced by one or another, which relates various choices together. Neither alternatives nor conflicts are subsets of each other. Decisions could also override one another, or can break down into other decisions or comprises. If a decision relationship does not fit into any of the above types, then the relating relationship can be used, but this is a weak relationship and is used primarily for documentation and illustrative reasons. The implication of relationships is that the decisions can now tell a story of the design process, bringing decision hierarchy and structure to the captured architectural knowledge.

CHAPTER 3

SYSTEM APPROACH TO DECISION CAPTURE AND EXPLORATION

The ability to represent software architectural design decisions would not be meaningful if there is no way to capture and explore decisions. Unfortunately, the capture and exploration of design decisions are closely tied and involve the idea of motivation. Software designers need to be motivated to capture their design decisions and one way to do this is to demonstrate the usefulness and the exploratory potential the decisions have to offer. However, the usefulness of the decisions depends on the acquisition of a set of design decisions; moreover, the designer may be required to capture even more information. The poor timing of the decision capture (Falessi *et al.*, 2008a, Grudin, 1996) also hinders motivation. Figure 1 illustrates this relationship between decision capture and decision exploration. To simplify the research, this double-spiral cycle will be addressed in a holistic manner while investigating design decision capture and exploration.

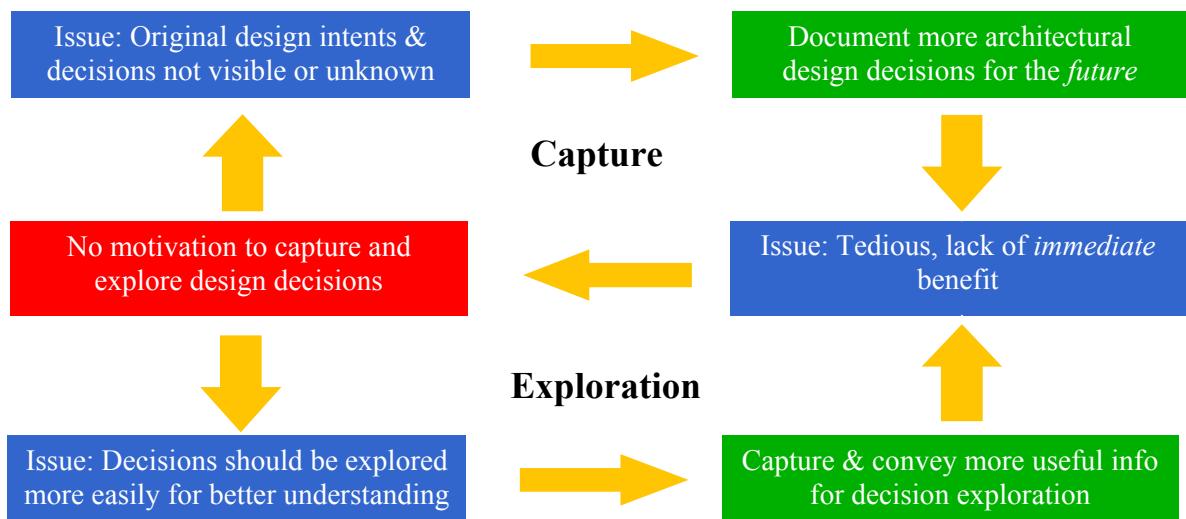


Figure 2: Decision capture and exploration relationship. The difficulty in capturing and using architectural design decisions involve the lack of motivation in capturing decisions in the present for future utilization. To end this cycle, we need to increase motivation for decision capture and exploration. We can achieve this by addressing two areas: improve the capture of decisions and increase the ability to explore design decisions.

Therefore, a systems-approach should be used to investigate architectural design decision capture and exploration. However, researchers have recognized that there are significant challenges to support the necessary functionality and architectural design decision use cases for design decision systems. Addressing all challenges and use cases is not possible in the scope of my research, so I will highlight a set of use cases and requirements for the system that will best address the challenges and functionality required in a design decision system.

3.1 Challenges and Requirements for Design Decision Systems

The challenges for software architectural design decision systems (some of which are shown in Table 10) often involve issues related to the software architecting/design process. Dueñas and Capilla summarize that after finding a design decision representation, the software architecting process is essentially a knowledge management process (Dueñas & Capilla, 2005), where the production of design and development artifacts is the result of applying the architectural and design knowledge during the design process. Moreover, as software development spans across a whole organization and involves many people, a system that manages architectural design decisions for a software project should be treated as a groupware system. Grudin discussed eight challenges that software developers need to address when developing groupware systems (Grudin, 1994). These challenges focus on how to increase a tool's usability in a group environment by implying that the success of a groupware tool depends on who benefits from the work and how well the tool supports a social or work process (including all the quirks and exceptions). We should develop decision systems with these challenges in mind. Essentially, good groupware tools promote less work and more benefit.

(J. Lee, 1997) identified seven issues for design rationale systems and also recognized that the person who bears the cost of capturing design decisions must be the person who benefits from decision capture, which echoes Grudin's first issue regarding the disparity between work and benefit. Lee's issues focus on how we can capture, access, and manage design decisions in addition to determining the uses and representation of design decisions. A key point is that design rationale systems (and design decision systems in general) should better support software design through dependency management, collaboration/project management, and design reuse/extension, while also recommending better maintenance, learning, and documentation

support. Lee also states that we can capture rationale through reconstruction, recordings, methodologies, and automatic generation. In addition, he describes that we must integrate captured design rationale across different users (and their viewpoints), different media (i.e., audio, video or text), and with various design modules/objects in other tools or processes.

Table 10: Design decision system issues and challenges

Topic (Source)	Issues and Challenges
Groupware Challenges (Grudin, 1994)	<ul style="list-style-type: none"> ▪ Disparity in work and benefit ▪ Critical mass ▪ Disruption of social processes ▪ Exception handling ▪ Unobtrusive accessibility ▪ Difficulty of evaluation ▪ Failure of intuition ▪ The adoption process
Issues for Design Rationale Systems (J. Lee, 1997)	<ul style="list-style-type: none"> ▪ What services to provide ▪ What to represent explicitly ▪ How to represent rationales ▪ How to produce rationales ▪ How to access rationales ▪ How to manage rationales ▪ How to integrate the system
Challenges for design rationale systems (Regli <i>et al.</i> , 2000)	<p><i>Technical challenges</i></p> <ul style="list-style-type: none"> ▪ Reducing the amount of knowledge workers within organizations to capture and manage design knowledge ▪ Making members of the organization aware of all relevant resources available to them, based on individual needs. ▪ Designing specific strategies for design rationale that will suit the needs of the organization (including reuse) <p><i>Design challenges</i></p> <ul style="list-style-type: none"> ▪ Using human-centered approaches ▪ Designing systems with identifiable benefits ▪ Supporting informal and formal knowledge ▪ Supporting multiple levels of content organization/design systems ▪ Building on a successful application as a best-practice ▪ Borrowing ideas from the field of participatory design, evolutionary growth, improvisational model, and Zimmermann and Selvin's framework (B. Zimmermann & Selvin, 1997).

A survey of design rationale systems performed by (Regli *et al.*, 2000) describes several technical and design challenges that we need to address for design rationale and decision systems. In analyzing their list of challenges, it appears that the overall technical challenge stems from the fact that we need to reduce the amount of design knowledge that people need to capture or manage. The fact that the design knowledge is highly dependent on the various needs of individuals or organizations makes this challenge more difficult to find a simple solution. Regli *et al.* also stated several design challenges for the design rationale systems. In general, using

human-centered approaches to design will allow the system to benefit the people who use the system. Moreover, we should design the systems in a way that will support both informal and formal knowledge capture, as well as support multiple levels of content organization so that the knowledge could be structured at any time using ways people can relate to and explore. These challenges and recommendations also are reminiscent of Grudin's groupware challenges, like the work/benefit mismatch for users and the leveraging of successful applications.

The lists of challenges and recommendations provide a foundation for the current development of requirements for architectural knowledge and design decision systems. These sets of requirements are natural extensions of the challenges and recommendations. Table 11 below shows a few of these requirement sets suggested by recent groups of researchers. Since we acknowledge that a design rationale (or an architectural decision support) system is a knowledge management system for the entire design process, the recommendations provided by (Regli *et al.*) make sense, as they address the capture and retrieval of design rationale with a knowledge management perspective. According to (Regli *et al.*), capturing design knowledge should be performed with minimal overhead and as little interference as possible on the natural progression of design activities, so that the designers can focus less on tedious documentation tasks and more on designing. The design rationale system should keep the knowledge consistent, including the support of conflict resolution when newly captured knowledge clashes with previously captured knowledge. For retrieval of design decisions, there should be strategies to retrieve large volumes of chronological data without causing people to navigate through all of the data. Query support should be implemented, which is useful to support various design tasks involving the browsing and viewing of the design knowledge.

Looking at design decisions from an architecture design perspective, (Dueñas & Capilla) lists five requirements decision support tools should have for the decision view of software architecture: multi-perspective support, visual representation, complexity control, groupware support, and the gradual formalization of design decisions. Multi-perspective support provides and highlights different facets of the decisions, depending on the particular person describing or viewing the decisions. Visual representation facilitates understanding and “replaying” of large sets of design decisions, of which the implementation of complexity control measures would

help with scalability and navigation of the decisions. Groupware support is inevitable as many people are involved in the design process. The last requirement is the gradual formalization of design decisions, in which it explicitly recognizes the fact that knowledge is often incomplete or difficult to express, so a knowledge creation process is highly recommended that will gradually build up sets of formalized design decisions.

Table 11: Requirements for architectural knowledge and design decision systems

Source	Requirements
Capture and Retrieval of Design Rationale (Regli <i>et al.</i> , 2000)	<p><i>Capture:</i></p> <ul style="list-style-type: none"> ▪ Capture process knowledge with minimal overhead and minimal interference with natural progression of design activities ▪ Resolve conflicts that arise when new knowledge clashes with previously captured knowledge ▪ Keep knowledge consistent <p><i>Retrieval:</i></p> <ul style="list-style-type: none"> ▪ Have retrieval strategies to manage large amounts of chronologically organized data ▪ Retrieve information without causing people to navigate through all of the data ▪ Support querying
Decision View Requirements (Dueñas & Capilla, 2005)	<ul style="list-style-type: none"> ▪ Multi-perspective support ▪ Visual representation ▪ Complexity control ▪ Groupware support ▪ Gradual formalization of design decisions
Effective tool support requirements for AK sharing (Farenhorst <i>et al.</i> , 2007)	<ul style="list-style-type: none"> ▪ Stakeholder-specific content ▪ Easy manipulation of content ▪ Descriptive in nature ▪ Support for AK codification ▪ Support for AK personalization ▪ Support for collaboration ▪ Sticky in nature

A recent study by (Farenhorst *et al.*) identifies requirements for tools that facilitate architectural knowledge sharing. In terms of content, architectural knowledge sharing tools should support stakeholder-specific content to address the various and customized needs of an organization in browsing and using the knowledge. Moreover, the tools should manipulate content easily, as changes to a software design are inevitable. The tools should naturally allow for descriptive perspectives on the content so they do not hinder the creativity of the designers, yet the tools should also support knowledge codification to allow formalized knowledge for knowledge retrieval and analysis. However, a degree of knowledge personalization is helpful as the less-structured form improves knowledge expression. If the personalized knowledge is collected and

shared in a collaborative environment, other designers can find out whom they can consult with by determining who is most knowledgeable in a particular area. The study also identified that the tools should be useful enough to encourage users to keep coming back to use the tools. Farenhorst describes this concept as a tool’s “stickiness” in which a tool would tend to stay attached to the user’s daily software design processes.

3.1.1 Visualization Tool Requirements

Dueñas and Capilla’s visual representation requirement of architectural design decisions launches into an entirely different area of research. The information visualization community is a large, long-established research community that attempts to explore how visualization can improve cognitive abilities to understand and identify high-level concepts with large sets of data (in the order of hundreds, tens of thousands, or often significantly more). The information visualization often has software support to visualize large sets of data. Conversely, the software community also has support from the visualization community to make sense of complex systems. Visualization helps with program comprehension and communicates information in ways the human mind can parse and understand. Various researchers have investigated how to improve the usability and effectiveness of visualization tools. Kienle and Müller summarized various works in visualization tools and came up with seven quality attributes and seven functional requirements shown in Table 12 that all visualization tools should have (Kienle & Müller, 2007).

Table 12: Visualization tool requirements (Kienle & Müller, 2007)

Quality Attributes	Functional Requirements
<ul style="list-style-type: none"> ▪ Rendering scalability ▪ Information scalability ▪ Interoperability ▪ Customizability ▪ Interactivity ▪ Usability ▪ Adoptability 	<ul style="list-style-type: none"> ▪ Views ▪ Abstraction ▪ Search ▪ Filters ▪ Code proximity ▪ Automatic layouts ▪ Undo/history

Summarizing Kienle and Müller’s work, the first two qualities of visualization tools refer to the scalability of the visualization tool. The visualization tool must be able to process information with reasonable performance when the dataset size is both large and small, whereas the second quality focuses on how much information to display on the screen to prevent overwhelming the

person viewing the data with large amounts of information. The visualization tools must be able to interoperate with other tools to promote information sharing and reuse functionality, while some form of customizability, like scripting, functional configuration files, or programmable interfaces, is useful to handle exception cases where a user may want to take a feature in a different direction than intended. Visualization tool interactivity gives the user control of the logic used to structure, navigate, and display information at the speed and direction of the user as a mental aid in exploratory applications like reverse engineering. The sixth quality a visualization tool should have is usability, but Kienle and Müller states that it is difficult to achieve, where the evaluation focuses more on the user interface and how we can reduce the obtrusiveness and cognitive overhead tied to the user interface. Finally, the adoptability of the tool will depend on how well it can support the needs of the users, such as customizability and functionality.

Since adoption depends on the functionality a tool should offer, Kienle and Müller summarizes seven functional requirements all visualization tools should have. Visualization tools should render information in the aspects of particular stakeholders to address the stakeholders' specific needs. Visualization tools should also support data abstraction, where low-level information can be generalized into groups or hierarchical structures to present and highlight new information not easily visible. To deal with potentially large amounts of data, visualization tools should use searching and filtering. Searching can help find a specific piece of information quickly while filtering helps reduce the amount of information shown to the user to help reduce information overload. In terms of visualization of software artifacts, code proximity is a way to improve program comprehension by linking visualization components as close to the relevant section in the software artifacts as possible. Code proximity helps users identify which area of the software code is represented by the visualization report. The sixth requirement is automatic layout of visualizations. This is essential, as it may not be feasible to manually filter, connect and disperse a large, complex dataset visually. The final requirement for visualization tools is the “undo” or “history” capability to allow users to revert to a previous state.

3.2 Use Cases for Design Decisions

After defining what a design decision system should have, we should define what activities a design decision system should support. Recalling the definition of architectural knowledge at the beginning of Chapter 2, design decisions are a subset of architectural knowledge, so we should find use cases for both the narrower-focused design decisions and the more general architectural knowledge. (For the scope of this section only, we will use architectural knowledge and design decisions interchangeably.) However, determining the use cases for architectural knowledge would require knowing who would use the knowledge and what they want to do with it.

3.2.1 Use Case Actors and Roles

Kruchten, Lago, van Vliet, and Wolf identified a list of actors which includes architects, developers, reviewers, analysts, maintainers, users and re-users of architectural knowledge, students, researchers and software tools (Kruchten *et al.*, 2005). Kruchten classifies the list of actors into two categories—active and passive, shown in Table 13. Active use case actors are producers of architectural knowledge, while the passive actors are the architectural knowledge consumers. Shortly afterwards, a group of researchers performed interviews with industry practitioners for their wish list on design decision uses (van der Ven *et al.*, 2006). The interviewed people, who are the use case actors, include architects, architecture reviewers, project managers, developers, and maintainers.

Comparing the two lists of actors (Table 14), Kruchten’s list is more specific, containing eleven classifications, while van der Ven’s list contains five. Although there are overlapping actors / roles (i.e., architects, reviewers, developers, and maintainers), van der Ven’s list contains the “project manager” role and Kruchten’s list includes the more general “users” and “re-users” of architectural knowledge in addition to the academic roles of “students” and “researchers”. The explicit “other architects” role in Kruchten’s list implies collaboration support with other architects in the same or different projects, and the addition of “analysts” and “software tools” would focus on the exploration and analysis of the captured design decisions to recommend or improve upon the software architecture the design decisions represent.

Table 13: Passive or active roles for architectural knowledge use case actors (Kruchten *et al.*, 2005)

Actors (roles)	Passive (consumers) / Active (producers)
Architects	Active
Other architects	Active *
Developers	Passive
Reviewers	Passive
Analysts	Passive *
Maintainers	Active *
Users	Passive *
Re-users	Active *
Students	Passive
Researchers	Passive *
Software tools	Active

** - The authors did not explicitly classify this role, so this classification is of my own opinion and not the authors'.*

Table 14: Use case actors for architectural knowledge and design decisions

Source	Actors (roles)
Architectural knowledge (AK) use case actors (Kruchten <i>et al.</i> , 2005)	Architects <i>Other architects</i> Developers Reviewers <i>Analysts</i> Maintainers <i>Users (of AK)</i> <i>Re-users (of AK)</i> Students Researchers Software tools
Design decision use case actors (van der Ven <i>et al.</i> , 2006)	Architect Architecture reviewer <i>Project manager</i> Developer Maintainer
Italicized roles are roles unique to each classification	

Interestingly, both use case actor lists did not fully address the generalized role of “stakeholder”. Stakeholders are people who have invested interest and resources in a project, and often have significant weight over the design and development of it. In general, stakeholders might include the owner, the client/customer, end-users, and the development organization. However, in contract-based software project, the stakeholders may agree on a high-level set of requirements for a software architect to base an architectural design on, so in the context of architectural knowledge, some stakeholders (like the client) have less influence on the architectural design

and hence they are left out of both use case lists. Moreover, we can also argue that the product manager who generated the list of requirements (and the architect to a lesser extent) usually represents the client, while the project manager could represent the development organization. In light of the above arguments, it is understandable why some actors did not make either list.

3.2.2 Use Cases

Using their list of actors, (Kruchten *et al.*, 2005) defined several use cases for architectural design decisions, listed in Table 15. Most of the use cases are self-explanatory and involve capturing, browsing and analyzing design decisions. “Spotting the subversive stakeholder” and “spotting the critical stakeholder” use cases are similar. However, they differ in that the former identifies people who could potentially affect the design significantly, while the latter focuses on how much a decision change would affect a particular stakeholder. The “integration” use case describes a situation where one needs to find an integration strategy to find how two or more systems can fit together.

After performing interviews and validating with industry practitioners, (van der Ven *et al.*) proposed a detailed use case model containing twenty-seven use cases, shown in Table 15. The model classifies the use cases by actor and goal levels in addition to the interdependencies between use cases, creating a grid they named the “knowledge grid”. When we compare the two lists of use-cases (see Table 16), we find that van der Ven’s list includes as well as extends most of Kruchten’s list, but van der Ven’s list does not explicitly address the “integration” use case.

Table 15: Architectural knowledge (design decision) use cases

Source	Requirements
Using architectural knowledge (Kruchten <i>et al.</i> , 2005)	<ul style="list-style-type: none"> ▪ Incremental architecture review ▪ Review for a specific concern ▪ Evaluate impact ▪ Get a rationale ▪ Study the chronology ▪ Add a decision ▪ Clean up the system ▪ Spot the subversive stakeholder ▪ Spot the critical stakeholder ▪ Clone architectural knowledge ▪ Integration ▪ Detection and interpretation of patterns
Using Architectural Decisions (van der Ven <i>et al.</i> , 2006)	<ul style="list-style-type: none"> ▪ 1. Check implementation against architectural decisions (needs #8) ▪ 2. Identify the subversive stakeholder (needs #3) ▪ 3. Identify key architectural decisions for a specific stakeholder (needs #1,9) ▪ 4. Perform a review for a specific concern (needs #3) ▪ 5. Check correctness (needs #8, 9) ▪ 6. Identify affected stakeholders on change (needs #3) ▪ 7. Identify unresolved concerns for a specific stakeholder (needs #9) ▪ 8. Keep up-to-date (needs #5) ▪ 9. Inform affected stakeholders (needs #5) ▪ 10. Retrieve an architectural decision (needs #6) ▪ 11. View the change of the architectural decisions over time (needs #5) ▪ 12. Add an architectural decision (needs #2) ▪ 13. Remove consequences of a cancelled architectural decision (needs #8) ▪ 14. Reuse architectural decisions (needs #14) ▪ 15. Recover architectural decisions (needs #6, 7) ▪ 16. Perform incremental architectural review (needs #1, 9) ▪ 17. Assess design maturity (needs #1) ▪ 18. Evaluate impact of an architectural decision ▪ 19. Evaluate consistency (needs #1) ▪ 20. Identify incompleteness (needs #1) ▪ 21. Conduct a risk analysis ▪ 22. Detect patterns of architectural decision dependencies ▪ 23. Check for superfluous architectural decisions ▪ 24. Cleanup the architecture ▪ 25. Conduct a trade-off analysis (needs #3) ▪ 26. Identify important architectural drivers (needs #3) ▪ 27. Get consequences of an architectural decision (needs #3, 6)

Table 16: Comparing Kruchten's list with van der Ven's list of use cases. The first two columns show how Kruchten's list of use cases could be covered by van der Ven's use cases. The third column assesses the implementation priority I assigned for this thesis.

Kruchten's list of use cases (Kruchten <i>et al.</i> , 2005)	Equivalent van der Ven's use case numbers (van der Ven <i>et al.</i> , 2006)	My Thesis Priority (lower # = higher priority)
Incremental architecture review	8, 11, 16	8 (Medium)
Review for a specific concern	4, 7, 18, 21, 25	4 (High)
Evaluate impact	1, 6, 9, 18, 25, 27	3 (High)
Get a rationale	5, 8, 10, 17	2 (High)
Study the chronology	1, 8, 11	7 (Medium)
Add a decision	12, 15	1 (High)
Clean up the system	5, 13, 19, 20, 23, 24	9 (Medium)
Spot the subversive stakeholder	2	5 (Medium)
Spot the critical stakeholder	2, 3	6 (Medium)
Clone architectural knowledge	14, 15	10 (Low)
Integration	—	12 (Low)
Detection and interpretation of patterns	17, 20, 22, 26	11 (Low)

3.3 Selecting the Use Cases

Although I have listed two sets of use cases for software architectural design decisions, one of the use case sets can be summarized by a general set (refer to Table 16). To simplify the discussions regarding the use cases in this thesis, I will use the more general use case set. One decision I needed to make about the chosen set of use cases is whether I would like a broad but shallow coverage of these use cases. A broad coverage would result in a wide sample of the utility of the design decisions, but it also hinders the study of how design decisions can be explored, as it requires detailed, in-depth implementations of the use cases. Unfortunately, limitations in time and resources prevent me from developing a design decision system (tool) with the complete implementation of all use cases. My thesis includes the exploration of design decisions and I have limited access to industry practitioners of various roles; therefore, I had to enforce a scope reduction for my research, which resulted in the implementation of a subset of the use cases.

The list of actors in Table 14 suggests the core actors the use cases should target should be the architect (and other architects), the architecture reviewer, the analyst, the project manager, the developer, and the maintainer. Referring to Table 13, the actors that are classified as “active” (architects, maintainers, re-users, and software tools) are producers of architectural knowledge. The most important by far is the architect’s role; without the architect, no relevant (or correct) knowledge about the architecture of a software project would be captured for analysis and study. The reviewer and the analyst would then be able to analyze and explore the captured knowledge and architectural design decisions to assess the state and structure of the architecture and can potentially cause more design decisions to be created or revised. These roles are the immediate knowledge consumers with significant influence; therefore, they are important roles that I must have for my tool. The project manager, developer, and maintainer are secondary knowledge consumers as they can exert some indirect influence on the architectural design.

In prioritizing the use cases according to complexity, actor availability for feedback, research goals, and interests, precedence goes first to the fulfillment of the basic decision capture and retrieval functionality, as these use cases are used by the most important role classifications (knowledge producer and the immediate knowledge consumers). Without the decision capture and retrieval support use cases, we cannot establish and manipulate a set of decisions to determine its exploratory potential. Next, the priority would go to decision impact analysis and concerns, as the immediate knowledge consumers find this beneficial and would motivate people to capture decisions. During the early stages of the study, I conversed with several software architects and developers in industry, and the feedback they provided supports this view.

The next priority group of use cases would be determining the relationships between the decision and the stakeholders, then on the effects of time on a set of design decisions, followed by how the design can be cleaned and improved. These use cases seem to offer more exploratory and analytical value with relatively less effort. The lowest priority level is the group of use cases that deal with multiple sets of design decisions; that is, the integration, cloning and pattern detection/interpretation. These use cases have not been investigated in detail because they require an established foundation for decision exploration beforehand. Investigating these use cases require significant design and implementation resources to be first spent on decision

exploration. Any automation of these use cases (even notification) requires significant algorithmic design or manipulation; therefore, these use cases would be put on the lowest priority. The comparison table that highlights the similarities between Kruchten and van der Ven's use case lists (Table 16) also shows the relative priority I assigned to each (or group of) use cases for the scope of this thesis.

3.4 Selecting the System Requirements

Proper implementation of the use cases of design decisions should follow the recommendations and requirements of a design decision system. Therefore, the developed decision support tool should follow the recommendations and requirements suggested by the various authors mentioned previously in Section 3.1.

The fundamental use cases of decision creation and retrieval ultimately depend on how we represent the design decisions, careful planning is necessary to select which decision representation model to use for the system. Details of the decision capture and representation requirements are found in Section 2.2. However, if we look at the capture and retrieval processes, the common theme of making decision capture, browsing, and manipulation easier is evident and is addressed by reducing the capture overhead, allowing more customizable or personalized content, gradually formalizing decisions, and improving the handling of large amounts of information (Dueñas & Capilla, 2005, Farenhorst *et al.*, 2007, Regli *et al.*, 2000). Other requirements of design decision systems are listed in Table 11 (see Section 3.1).

The decision view requirements proposed by Dueñas and Capilla is a mandatory set of requirements to fulfil, as it summarizes many requirements well. A design decision system should support multiple perspectives to handle various stakeholder needs and documentation biases while acknowledging the collaborative nature of design by requiring groupware support. The decision view also requires complexity control and gradual decision formalization to handle the large volume of information to capture, browse, and manipulate; furthermore, the visual representation requirement could significantly aid in those areas as well. Though second in priority, implementing query support, consistency checks, and conflict management (Regli *et al.*, 2000) as well as implementing measures to increase the system's "stickiness" (Farenhorst *et al.*,

2007) are highly preferred and the system would meet all the suggested requirements. I identified that the adoptability and usability visualization tool requirements are difficult requirements to satisfy, since they involve in-depth study of the way people use the tools and require additional studies to be performed. As I acknowledge my limited time and resources, I address these two requirements less significantly than the other requirements in the initial implementation of the tool. Later tool design iterations will focus on these two requirements. To summarize, Table 17 below gathers together all the selected requirements and use cases I intend to implement in my software architectural design decision support tool. The rest of the thesis references the requirement and use case identifiers used in this table.

Table 17: Summary of selected requirements and use cases

ID	Requirement/Use Case	Source
R1	Capture with minimal overhead	(Regli <i>et al.</i> , 2000)
R2	Resolve conflicts	(<i>Ibid.</i>)
R3	Knowledge consistency	(<i>Ibid.</i>)
R4	Retrieval strategies to manage large datasets	(<i>Ibid.</i>)
R5	Retrieve information without navigating through all data	(<i>Ibid.</i>)
R6	Support querying	(<i>Ibid.</i>)
R7	Multi-perspective	(Dueñas & Capilla, 2005)
R8	Visual representation	(<i>Ibid.</i>)
R9	Complexity control	(<i>Ibid.</i>)
R10	Groupware / Collaboration	(<i>Ibid.</i>)/(Farenhorst <i>et al.</i> , 2007)
R11	Gradual decision formalization	(Dueñas & Capilla, 2005)
R12	Stakeholder-specific content	(Farenhorst <i>et al.</i> , 2007)
R13	Easy content manipulation	(<i>Ibid.</i>)
R14	Descriptive in nature	(<i>Ibid.</i>)
R15	Knowledge codification	(<i>Ibid.</i>)
R16	Knowledge personalization	(<i>Ibid.</i>)
R17	Sticky in nature	(<i>Ibid.</i>)
U1	Incremental architecture review	(Kruchten <i>et al.</i> , 2005)
U2	Review for a specific concern	(<i>Ibid.</i>)
U3	Evaluate impact	(<i>Ibid.</i>)
U4	Get a rationale	(<i>Ibid.</i>)
U5	Study the chronology	(<i>Ibid.</i>)
U6	Add a decision	(<i>Ibid.</i>)
U7	Clean up the system	(<i>Ibid.</i>)
U8	Spot the subversive stakeholder	(<i>Ibid.</i>)
U9	Spot the critical stakeholder	(<i>Ibid.</i>)
U10	Clone architectural knowledge	(<i>Ibid.</i>)
U11	Integration	(<i>Ibid.</i>)
U12	Detection and interpretation of patterns	(<i>Ibid.</i>)

ID	Requirement/Use Case	Source
V1	Rendering scalability	(Kienle & Müller, 2007)
V2	Information scalability	(<i>Ibid.</i>)
V3	Interoperability	(<i>Ibid.</i>)
V4	Customizability	(<i>Ibid.</i>)
V5	Interactivity	(<i>Ibid.</i>)
V6	Usability	(<i>Ibid.</i>)
V7	Adoptability	(<i>Ibid.</i>)
V8	Views	(<i>Ibid.</i>)
V9	Abstraction	(<i>Ibid.</i>)
V10	Search (Query)	(<i>Ibid.</i>)
V11	Filters	(<i>Ibid.</i>)
V12	Code proximity	(<i>Ibid.</i>)
V13	Automatic layouts	(<i>Ibid.</i>)

3.5 Meeting some Challenges

We can check the selected requirements and use cases against the challenges for design decision systems (found in Table 10). Looking at Lee’s issues for design rationale systems, we have addressed the issues of what services to provide through the selection of use cases (refer to Section 3.3), as well as the “what” and “how” to represent the design decisions through the decision representation discussion (refer to Chapter 2). Minimizing the capture overhead and supporting custom or organization-specific processes would address the decision production issue. Complexity control measures and visualization requirements are approaches to the decision access and management issues. Integration across various mediums, representations, and systems remains a significant challenge, but unifying the various requirements together and developing a multiplatform tool would attempt to address this.

Likewise, increasing the immediate benefit to the decision capturer, reducing the interference of the decision capture or retrieval processes and supporting custom or organization-specific processes would address most of Grudin’s eight challenges. If the focus is on providing immediate benefit to the capturer, there will less work/benefit disparity so that collaborative features become extensions of the immediate benefit and not an additional chore. These points also summarize Regli’s technical challenges. Combined with the requirement for gradual formalization of decisions and avoiding new tools (or processes) that the organizations are not familiar with, we can also address Regli’s design challenges as well.

CHAPTER 4

DECISION CAPTURE AND VISUALIZATION SUPPORT

The success of a design decision system depends not only on the challenges the system should address and the set of requirements needed to implement such a system, it also depends on how the system will be implemented *within* an organization. In the previous chapter, we have looked at the challenges and requirements of a system-based approach for architectural design decision capture and exploration. We have also looked at what we can do with a set of design decisions and what use cases such a design decision system must support. Furthermore, we reflected that software development involves many people, so any software architectural design decision system would need to follow Grudin's groupware challenges. One major challenge and a critical element of every system is how a software system can be adopted and used by people within an organization.

This challenge can be described using an analogy of a corkboard in an office. This analogy starts off with many office workers complaining that they cannot easily notify each other about events, share anecdotes, or post pictures and humorous comic strips from the daily newspapers with the entire office. Their current system of using broadcast e-mails resulted in everyone's mail boxes being flooded with e-mail. To remedy the situation, the office acquired a large corkboard with many features, including a magnetic plate for photos, two bright lamps, a whiteboard, and markers of different colours and sizes. Clearly, this corkboard addressed the needs of the office workers. However, the corkboard was quite large and they could neither fit it on the wall in the kitchen nor by the water cooler. They were able to find enough wall space in a dark hallway near an emergency exit. Unfortunately, the office workers did not often walk by the emergency exit, and the hallway had no electrical outlets, rendering the two lamps useless. Pictures posted there were difficult to see, only two marker colours were distinguishable in the dim light, they had no magnets, and messages left there expire unnoticed. Most of the notices on the bulletin board were e-mail messages that the office workers manually printed and posted. It

was such an inconvenience to use the corkboard that, after several months, the office workers resorted back to their old, system of broadcasting e-mails and the corkboard went unused. The corkboard with its many features failed because it simply did not fit into the daily routines and information flow at the office.

Applying this corkboard analogy to software systems, a software system should address the specific needs of the organization without hindering the organization's daily routines and work flow. In other words, a software system is not just a tool to get work done; its use has to fit the work activities and processes. If an organization makes a product C by first making A and then making B, then any tool to improve the production of C should improve the production of A and B and avoid introducing X, Y, or Z into the process, even if it does improve C slightly in the end. Any additional work or complication must be justified by significant benefit. For marginal amounts of benefit, it is better to break down a step into smaller parts (A becomes A' and A'') and improving each smaller part than adding new parts.

In this chapter, I propose that decision capture can be improved by using three decision capture approaches to better meet the immediate needs of the people capturing the design decisions. As well, I propose four aspects of decision visualization to better implement the requirement of visualization in the context of decision capture and exploration so that people can better understand the captured architectural design decisions. The proposed approaches and aspects are measured against the selected requirements summarized in Table 17.

4.1 Decision Capture

In terms of the decision capturing process within an organization, the importance and method of capturing decisions can vary depending on the type, size, and risk of the project being developed. Small projects, such as websites or utility tools, may not warrant the amount of effort needed to capture architectural knowledge, so decision documentation is unnecessary. Projects involving software with a long service life-span may require significant documentation for code maintenance and evolution; moreover, large or high-risk projects, like safety-critical systems, require careful planning, accurate documentation, and extensive reviews in both documentation and implementation to ensure that the right decisions and implementations are made.

Furthermore, the development state of the project affects how decisions are captured. Numerous architectural decisions are made during the early stages of design but many of these decisions are vague ideas or are tentative and do not make it past the later stages of design. Capturing decisions in the mature stages of development mean that the majority of the architectural decisions are made already and are highly specific, but many of these decisions are forgotten before they could be documented. Decisions should be captured during the early stages of design before they are forgotten, while decisions should be captured during the later stages of design when the decisions are concrete and specific. Clearly, different situations require different capturing approaches to address the specific needs of both the organization and the situation.

4.1.1 Approaches to Decision Capture

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. I propose three approaches to decision capturing. These three approaches are formal elicitation, lightweight top-down capture, and lightweight bottom-up capture. Each approach takes a different perspective to decision capture (requirement R7) to address the various decision capture needs of an organization. To demonstrate each approach, I suggest particular methods that implement these approaches and I describe the steps of these methods.

4.1.1.1 Formal Elicitation

Formal elicitation of software architectural design decisions is the method of gathering software decisions in an explicit and structured manner. This is normally performed in several long sessions devoted for this purpose. The approach, as illustrated in Figure 3, may be better described as the “Big Bang” decision capturing approach, because the tacit decisions are materialized and made both explicit and formal without any intermediary steps. Articulation (as Nonaka puts it) in a single step is called “elicitation” in this capturing approach.

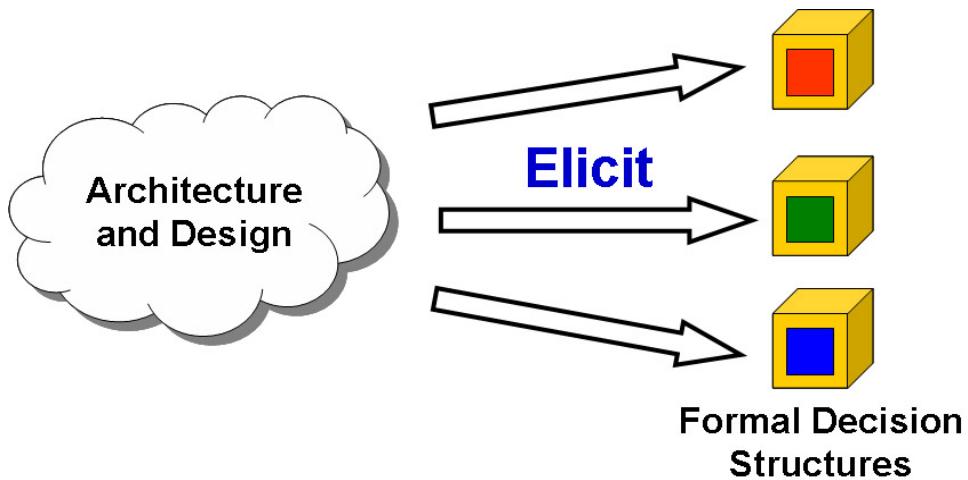


Figure 3: Generalization of formal elicitation. More easily understood as the “Big Bang” decision capture approach, tacit decisions are articulated (elicited) and made explicit by forming decision structures using a particular decision representation model. The blocks represent these decision structures, and the solid colour inside each block simply denotes different decisions. Figure from (L. Lee & Kruchten, 2008a) with permission from IEEE.

In this approach, decisions are elicited directly or after-the-fact, with an emphasis on gathering detailed information on decisions. The decision information is structured formally using a particular decision representation model. This model can be a design rationale model or a decision entity model, and it guides decision capturers to document decisions with sufficient information, such as the issues, alternatives, choices, and rationale that were present when the decision was made. These details help make the captured design decisions more self-contained so that someone new to the software system can quickly understand the nature of the design through the decisions’ context. When capturing the design decisions (use case U6), the details would help with query support (requirement R6) by providing a large base of information, and the formalized approach helps with knowledge codification (requirement R15) requirements for knowledge consistency (R3) by providing sufficient amount of information. 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 because the process prompts the decision capturers to enter specific decision information in a consistent and predictable manner to systematically create a decision that could follow one of many design

rationale and decision representation models mentioned above. Since this approach is by definition a single-step approach, it is obvious that there is only one type of method that would implement this approach. This method is simply “elicitation”.

4.1.1.2 Lightweight Top-Down Capture

To complement the formal capturing approach mentioned above, I 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. This ability addresses the requirement to capture with minimal overhead (requirement R1) and the gradual formalization of decisions (requirement R11). A method that implements this splits the formal elicitation approach into three steps: flag, filter, and form (L. Lee & Kruchten, 2007). This method is illustrated in Figure 4.

Flagging

Flagging is the capture of candidate decisions from the source in which they are found. Sources of decision inspirations may be from magazine articles, books, audio/video recordings, e-mails, electronic documents, or internet web pages. Candidate decisions are ones that are considered, but not necessary for a design. Using a reference marker, which briefly describes the essence of the decision and points to the source where it is found, can capture a decision candidate. Flagged candidate decisions are called decision references. Flagging can be performed with little worry over the immediate relevance or priority of the decisions as the sorting tasks can be performed at a later point in time during the filtering step. In this manner, the captured knowledge can be personalized to the design decision capturer and other stakeholders (requirements R12).

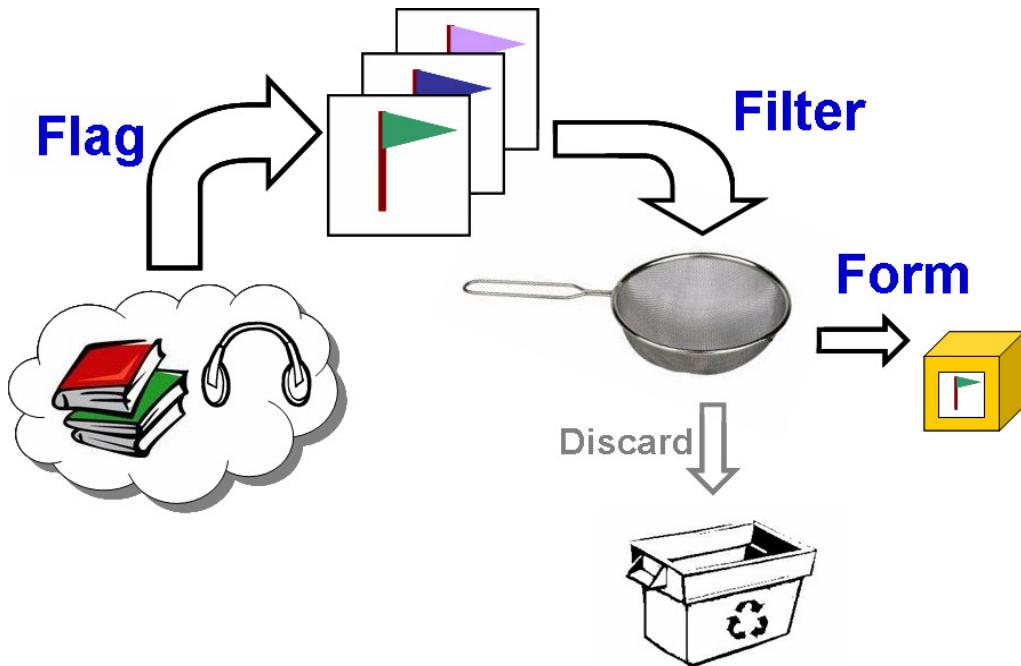


Figure 4: A lightweight top-down capture method. This method implements the lightweight top-down capture by breaking the decision capture process into three steps: flag, filter, and form. Decision inspiration found in various media such as books, documents, recordings, e-mails, or meeting minutes can be flagged and stored in a list of candidate decision references (“candidate” because they are not full decisions yet) for future retrieval. The list can be scanned at a later point in time in which the candidate decision references are filtered for relevance (represented by the sieve). Decision references that are still relevant can be formed into formal decision structures (represented by the block). Decisions that are no longer relevant are discarded, but kept handy in a repository (represented by the recycle bin) in case we need to find alternate decisions. Figure from (L. Lee & Kruchten, 2008a) with permission from IEEE.

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. This step allows stakeholder-specific content to be captured (requirement R12). The filtering step confirms and promotes the selected decision candidates to be lightweight versions of the full, formal design decisions.

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. This would complete the requirement to gradually formalize design decisions with minimal overhead (requirements R1, R11, and R15). 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. This step is similar to the creation of formal decision structures in the formal elicitation approach, except that the decision capturer has intermediary information (and additional information captured, like decision sources) to help flesh out the decision details. This intermediary information is personalized and stakeholder-specific knowledge that the decision capturers find relevant (requirements R12 and R16).

4.1.1.3 Lightweight Bottom-Up Capture

With the lightweight decision capture from the software programmer's perspective in mind, I propose a second new capturing method that supplements both the formal capturing methods and the lightweight top-down method described above by specifically addressing decision capture in the mature development and maintenance phases of a software project. 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 (requirement R11). 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 (requirement R12). Figure 5 illustrates the bottom-up capture method as applied to software source code. Filtering decisions for relevance is not necessary for bottom-up capture because fewer decisions are made when the design matures and development progresses; furthermore, architectural decisions at this point in time are often concrete and to-the-point, usually addressing a specific architectural issue.

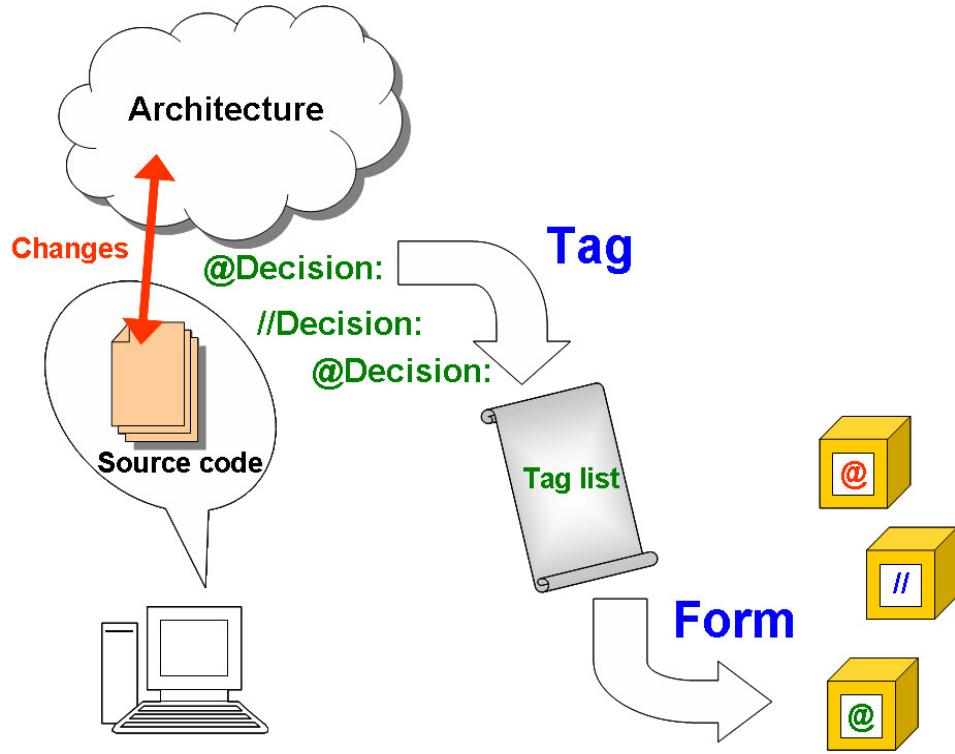


Figure 5: A lightweight bottom-up capture method. Similar to the flag-filter-form method, this lightweight bottom-up method divides the decision capture process into two steps, tag and form. This figure shows the method as it applies to software source code. During the later stages of development and code maintenance, architectural changes are often reflected in the source code, like workarounds and patches. A software developer can document (tag) architectural decisions close to the affected areas of code using a decision tags or code comments (such as @decision or //decision). The list of tags is stored within the source code, and can be displayed to developers during peer code review or code commit to form a formal decision structure (represented by the blocks). Figure from (L. Lee & Kruchten, 2008a) with permission from IEEE.

Tagging

I define the verb, “tag”, to mean the act of attaching small amounts of specific information onto an article or other objects for later information retrieval regarding it. In the context of architectural design decisions, 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. The tagging step captures decisions without significant documentation effort (requirement R1) by capturing

decisions as close to the artifacts that they are most concerned with. The captured decisions would be highly relevant and specific to the capturer (requirements R12 and R16). Software architects and other designers could document their decisions without leaving the design tools they use most and are most familiar with (requirement R17). For example, a software architect is studying a large class diagram and identified a collection of tightly coupled classes that heavily depend on each other. The architect feels that these coupled classes need refactoring and the architect would then tag the collection of classes within the class diagram with his decision and a supporting reason. The architect could also tag his decision on the class diagram after refactoring the classes. In general, the primary difference between tagging and flagging is that flagging documents decision information in an *external* repository, while tagging documents decision information *within* the object in concern.

In the context of software code, tagging is a common term to describe the act of placing identifiers (like “@Decision” or “//Decision” code comments) 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 deal with the particular design section represented in the artifact. A related work uses tagging and code commenting as “waypoints” to document thought flow and code navigation (Storey *et al.*, 2006). Other related works include social tagging applications like Delicious (<http://del.icio.us>) for web links and Flickr (<http://flickr.com>) for photographs. Broadly speaking, social tagging usually stores tags externally (similar to web-bookmarks and decision flagging), while decision tagging stores decisions inside the design artifacts of concern.

Forming

The decisions tagged within source code are typically succinct 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. As software artifacts like class diagrams and source code could change often, formalization of the decisions (requirement R15) is important to support knowledge consistency and awareness (requirement R3).

4.1.2 Customized Decision Capture

Multiple decision capturing approaches allow organizations to choose a better approach for their needs. The three decision capturing approaches I propose give organizations more flexibility in how they capture decisions. 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. Moreover, the methods that implement each approach can be further customized to better fit the needs of the organization.

4.1.2.1 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 2-4 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 were captured using the formal elicitation approach. Thus the two lightweight approaches implicitly support the knowledge codification requirement (requirement R15). 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. The flexibility of the source, form, and manner of documenting the design decisions satisfies the requirement to be descriptive in nature (requirement R14). 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 using a particular approach can be advantageous 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 but not documented, such as during post-implementation reverse engineering, design comprehension and documentation. Moreover, this capture approach is sometimes preferred because of its simplicity and shortest turnaround time before return on investment — the single-step capturing approach is direct so the decisions can be explored in great detail immediately after creation. 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 (Tang *et al.*, 2006). Formal elicitation does support gradual decision formalization (requirement R11) implicitly in that design decision information can be captured at various points in time. But the act of going back to an unfinished design decision could be difficult; from personal experience, things left incomplete tend to stay incomplete. Therefore, we need to have some guidance or assistance in capturing decisions in situations where the design knowledge is intentionally or unavoidably left incomplete.

The two lightweight approaches focus on the Gestalt, in which the incomplete nature of knowledge can be brought together to form a whole picture. Specifically, the two approaches are designed to facilitate and support the design activities of the software architects, designers, and the rest of the software development organization by explicitly documenting smaller pieces of information so that their accumulation would describe a designer's decisions and knowledge as a whole. Furthermore, the focus on lightweight, incremental decision capture reduces the impact of the decision capturing process on an organization's design activities. Top-down capture of design decisions addresses decision making in early design stages, when 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 determine the path of progression for the design. The bottom-up approach is suitable 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 documentation of the technical design issues related to the architecture. Moreover, the resulting documented decision can also be easily referenced throughout the lifetime of the design artifact because it is a part of the product.

4.1.2.2 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, I mentioned the knowledge needs of an organization. Each capture approach satisfies a certain design process perspective and each method implementation can be customized to adapt to varying capture goals.

For some organizations, using a portion of a lightweight method would suffice; just capturing decision references in its unrefined state is sufficient 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 (Gotel & Finkelstein, 1995), 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.

There is also the concern of personal and proprietary decision disclosure. People often prefer to keep personal decisions private. Some decisions are made with personal or political motives, and are rarely documented under fear of their discovery. For example, several employees working for a company in times of economic uncertainty would conceal some of their design knowledge in hopes to become indispensable and thus attain some level of employment security. In another case, a joint venture with another company on a project could involve varying levels of design disclosure. In either case, the designers would soon forget their original intents and decisions if decisions were not documented at all under fear of their discovery. Varying levels of disclosure

at the discretion of the organization and the decision capturers would alleviate this fear and promote documentation of sensitive decisions. Depending on the organization, publicity levels can be enforced. Every person capturing his or her design decisions gets a private area to store their personal decisions. Decisions can be assigned a publicity level, such as “personal” or “organization-wide”, at any point in time. This concept enables selective-release of design decisions, where decisions can be selectively shared with other people or the rest of the organization. Support for the selective-release of design decisions is provided by the “publicity level” attribute of the design decision representation model described in Section 2.5.

4.2 Decision Visualization

The formalized structure of explicit design decision representation in software architecture offers high decision analysis and exploration potential. However, the analytical and explorative capabilities of architectural design decision representation are bound by the way the information is presented. In the previous chapter, I make a special note that visual representation is one of Dueñas and Capilla’s requirements for the design decision view of software architecture (requirement R8). Design decision visualization facilitates easier understanding of the architecture and provides a better walkthrough of the designer’s decisions and intents because it is capable of retrieving and displaying large sets of data in a meaningful way without overwhelming the people viewing it (requirements R4 and R5). A design decision system should support visualization as an integral part of the system. In focusing on the exploration of architectural design decisions, the visualizations should also support the architectural knowledge and design decision use cases outlined in Chapter 3 to help people perform their decision-related tasks more effectively. To arrive at a visualization solution for software architectural design decisions, one needs to know of the various visualization techniques currently available, and identify which aspects of design decisions to visualize, guided by the use cases, which are how design decisions can be used.

4.2.1 Visualization and Design Decisions

The information visualization community dedicate their research to help people perform specific tasks more effectively by improving the communication and cognition of a large set of complex or abstract information through visual representations. As information comes in many forms,

there are also many ways to represent information visually. Information in the form of text can be arranged in paragraphs, lists, or tables, while numeric or relational information can be represented using shapes, graphs, or hierarchical structures. In general, there are three types of information: ordinal, nominal, and quantitative. Quantitative information has a magnitude and can be measured. Ordinal information has an established order but may not have a magnitude, such as information based on rankings, time, or sequence. Nominal information is qualitative or descriptive, such as texture, shape, or name. Some types of information (like colour) may fit into multiple categories depending on the context or representation. Charts and plot-graphs are useful to compare quantitative information against other quantitative or ordinal information, such as the number of decisions made over a period of time. Nominal information can be plotted against quantity, such as the frequency of occurrence of the word “the” in a literary work. Relationships and associations between nominal information can also be represented graphically using nodes and edges.

In a strict sense, people have used visual representations specifically to understand large amounts of information since the late 1700’s (Heer *et al.*, 2005). Many new information visualization techniques have been investigated since then, such as graph drawings, tree mappings, clustering, cloud representation, bundling, and metaphor representations. Graphs (nodes and edges) are useful to display associations, hierarchies and dependencies while treemaps are useful to illustrate hierarchies based on subsets. Treemaps, (as well as clustering, bundling, and cloud representation) help people identify and group information based on outliers and commonalities (Munzner, 2000). Metaphor representations allow differences and anomalies to be detected based on familiarity. Further visualization techniques to help people navigate, understand and manipulate large amounts of information within a single view of the include animation and interactivity and navigation, spatial distortion (like “fisheye” or hyperbolic graphs), colour, dimensionality, and information compressibility.

The software maintenance and program comprehension communities applied many of these visualization concepts to better understand the software in terms of the software structure, behaviour or evolution. A recent software visualization workshop featured papers that visualize the sequence of method calls within a program as graph (Deelen *et al.*, 2007), a non-linear

timeline of dynamic memory allocations (Moreta & Telea, 2007), city-block/building metaphor of class packages and classes using three-dimensional treemaps (Wettel & Lanza, 2007) and a edge-bundling graph of a program's execution trace (Holten *et al.*, 2007). There is also a visualization tool named SoftArchViz (Sawant & Bali, 2007) that appears to be closely related to software design, but upon closer analysis it is actually a tool that visualizes the implemented software architecture through a component-connector view of software classes, member variables, logical structuring (file system and packaging), number of threads, functions, and the distance away from the hardware level of abstraction. However, there is little work in visualizing the architecture as a set of design decisions.

Determining how software architectural design decisions should be visualized is difficult because design decisions have many ordinal, quantitative, and nominal attributes. (Ordinal attributes include the decision creation/modification time, state, disclosure level, and relationship strength. Quantitative attributes include the number of changes and relationships. Nominal attributes include the keywords, relationship type, categories, source, and author of the decisions.) Depending on the context and situation, certain attributes are compared or evaluated more frequently. For example, when finding a subversive stakeholder (use case U8), the author, timestamp, and change log of a design decision are more important than when understanding or reviewing the decisions behind the architecture, which significantly involve the decision rationale and relationships. As decision visualization should help people perform their tasks better, we should have special visualizations that focus on certain aspects of design decisions to reduce visualization complexity and improve the task assistance.

4.2.2 Essential Decision Visualization Aspects

I propose four visualization aspects (L. Lee & Kruchten, 2008c) that should be addressed when visualizing design decisions: these four aspects are tabular lists, graphical structure visualization, chronology visualization and decision impact visualization. These aspects abstract and represent the decision representation model visually to address the visual representation requirement (requirement R8) and contribute to the complexity control requirement (requirement R9). The four visualization aspects also support the multiple-perspective approach to address specific

situations and foci (requirement R7). These situations involve the twelve use cases of software architectural design decisions and will be discussed in the following subsection.

4.2.2.1 Tabular Lists

The purpose of this visualization aspect is to supply a quick and effective way to browse and retrieve information from design decisions (requirements R4 and R5). The textual tabular representation facilitates decision querying and simple decision entry and manipulation (requirements R1 and R13) because the data representation can be easily parsed on a computer screen or on a paper printout. Although tables provide efficient textual display of decision information, it is difficult to quickly trace and assess decision structures, relationships, and properties when the decision set becomes large or changes relationships frequently. Although there is framework support for sorting, filtering and querying (requirements R6, V10 and V11), a better retrieval strategy for large datasets is needed to adequately support requirements R4 and R5 and another visualization aspect is needed to better handle the complexity (requirement R6).

4.2.2.2 Graphical Structure Visualization

The goal of this aspect is to increase understanding of the architecture's decision structure. The decision structure guides the capture, perusal, and manipulation of decisions and their relationships without sacrificing the comprehension of the structure of the architecture for managing the design decisions, especially when the decision sets become large. An effective way to sort and analyze large sets of decision information is to graphically represent the decisions (requirements R4, R5, and R8). Graphs are used to visualize decisions: decisions are represented as nodes and the relationships are the edges. The visualization should be able to display decisions, their attributes, and their relationships to each other separately from the architectural components, or, in other words, a decision-only view of the software architecture.

A significant benefit for graphical structure visualization is its cognitive assistance, which helps individuals to create a mental map of the decisions. Other benefits include the ability to detect missing or orphaned decisions that may denote design incompleteness and the preservation of decision contexts in relation to one another. These benefits support the requirements to resolve conflicts and maintain knowledge consistency (requirements R2 and R3).

Decision relationships are better represented visually using decision graphs than lists in a table, despite the fact that tables provide a more efficient textual display of decision information. The emphasis on a mutable, visual manipulation interface complements the dynamic nature of the decisions and satisfies the requirement for easy content manipulation (requirement R13). A designer should be able to create and manage decision relationships easily, such as drawing a line or dragging one decision on top of another. Likewise, changing a decision's attributes can be made easier by selecting decisions through the visual interface.

4.2.2.3 Chronology Visualization

The goal of this aspect is to increase understanding of the architecture's dynamic nature. Software design changes over a period of time, so the design decisions made will also change. The visualization should handle the evolution of the design decisions and should support versioning and the state of the decisions. The decision state can change at any time, implying that any decision related to a mature decision could also be affected when that mature decision becomes obsolete.

Keeping track of the history of the changes would better explain the architectural story and reasons behind the design when we study the decision chronology (use case U5). Moreover, a timeline view is suggested that will display decisions that were created or modified during a specific time interval. This would be beneficial in periodic design reviews, where the reviewers can find what has changed since the last review or determine the design maturity from the decisions. The chronology visualization aspect supports certain query types and filtering (requirements R6 and V10) against time or author to determine what decisions were changed recently and by whom. This aspect provides a direct way to view and assess the gradual formalization of design decisions by studying the various decision versions over a period of time (requirement R11). Easier decision manipulation (requirement R13) is made possible because this visualization aspect reduces the amount of information a user needs to sift through or modify (requirement R5) as a part of the chronology retrieval strategy for decisions (requirement R4).

4.2.2.4 Impact Visualization

The goal of this aspect is to increase the understanding of the architecture's dependencies on its set of design decisions. This visualization helps visually identify the impact decisions have on

each other using the decision relationships and properties as well as links to software artifacts and architectural components. The purpose is to assist in finding and resolving decision conflicts in addition to maintaining knowledge consistency (requirements R2 and R3). To be more concise, the decision impact visualization aspect utilizes the traceability provided by the artifact support and the decision attributes represented in the structural support to create a potential impact matrix for software architects, designers, and developers to draw conclusions upon. This matrix represents a large volume of information and browsing it can be overwhelming. Focussing on certain categories of the matrix helps reduce the amount of information during retrieval and browsing (requirements R4 and R5). The visualization of this matrix provides an entry point into decision exploration and analysis by linking potentially impacting decisions together and making the impact relationships obvious in the visualization. Identified impacted decisions can be easily viewed and manipulated from this aspect (requirement R13). An example of a decision impact matrix is shown in Table 18.

Table 18: Decision impact matrix example. Short-hand notation and abbreviations are used to keep the table tidy.

Design Decision	Relationship with Decision				Decision Attributes							
	I	II	III	IV	Epitome	Rationale	Scope	Category	Author	State	Publicity	Source
I *	-	cs	cf		Use dot NET 3.5	Acquired technology uses dot NET 3.5	Back-server	Framework, Back end	LL	Decided	Organiz'n	Acquis'n tech doc , p. 143, sec 9.12
II *		-	cp		Deploy on multiple platforms	20% of market not using Windows	Client agent, back server	Deployment, Agent, market needs	JW	Apprv'd	Public	Product brochure for ver. 1.5 & up
III *	cf		-		Support IBM OS/2	Legacy support	Client agent	Legacy, Agent, Compatible	LL	Chllng'd	Organiz'n	SC from AIE R&D
IV *		o	f	-	Support popular configurations only	Lack of resources in dev, short time to market	Dev., market time, System.	Deployment, testing, market needs Executive decision	MTW	Tent've	Personal	LL, from Jun 08 IT Monthly magazine
Relationships: (In the form "<Decision *> [relates to] <Decision>")												
cs = constrain f = forbid o = override cp = comprise of				en = enable b = bound to				s = subsume a = alternative to			cf = conflict with r = related to	

4.2.3 Visualization and Use Cases

The four visualization aspects are designed to support the various use cases of design decisions discussed in section 3.2. These use cases unite the set of requirements together to address how we can use visualization to explore design decisions. An important idea is that each of the four

visualization aspects focuses on certain aspects and situations represented by the use cases. A handful of visualization aspects cannot adequately view and model all known aspects of design decisions to satisfy the needs of every designer or developer in the same way that we cannot directly view all sides of a three-dimensional object using a single spatial-perspective. Thus, each visualization aspect will attempt to address particular use cases so that collectively, all the use cases can be met. This is shown in Table 19.

Table 19: Use cases and the four decision visualization aspects

Design Decision Use cases (Kruchten et al., 2005)	Decision Visualization Aspects			
	Tabular list	Graph'l structure	Chronology	Impact
U1: Incremental architecture review	—	—	Supported	—
U2: Review for a specific concern	Supported	Supported	Supported	Supported
U3: Evaluate impact	—	Supported	—	Supported
U4: Get a rationale	Supported	Supported	Supported	Supported
U5: Study the chronology	—	—	Supported	—
U6: Add a decision	Supported	Supported	—	—
U7: Clean up the system	Supported	Supported	—	Supported
U8: Spot the subversive stakeholder	—	—	Supported	Supported
U9: Spot the critical stakeholder	—	—	Supported	Supported
U10: Clone architectural knowledge	Supported	Supported	Supported	Supported
U11: Integration	Supported	Supported	—	—
U12: Detection & interpretation of patterns	—	Supported	—	Supported

The four most important use cases to fulfill is adding design decisions, getting the rationale behind a decision, evaluating the impact of a decision, and reviewing the decisions for a specific architectural concern. As decisions are created and manipulated in an environment where people can quickly browse and understand the structure of decisions, I support decision capture (use case U6) in the tabular listing and the graphical structure visualization aspects. The decision impact visualization aspect is designed specifically to evaluate the potential impact if a decision is changed or removed (use case U3). Decision impact analysis is an important decision use case, where six of (van der Ven et al.)'s twenty-seven use cases address it (refer back to Table 16), and the feedback from software architects and designers in industry supports its importance. In all four decision visualization aspects, the user can view the decision rationale and other details (use case U4) simply by selecting the decision and getting the rationale stored within it.

The next four important use cases surround the changes made to the system being designed. These use cases are spotting the subversive stakeholders, spotting the critical stakeholders, studying the chronology, and performing an incremental architecture review (use cases U1, U5, U8, and U9). This is achieved through the decision chronology view, which looks at the decisions created or modified during a specific time interval and maps them to a timeline. The result of this aspect allows users to find what has changed since the last review and which decisions are being modified. This view also makes it easier to find which stakeholders are making the changes (i.e., subversive stakeholders) and which stakeholders would be most affected by a decision change (i.e., critical stakeholders).

The final set of use cases deal with the decision maintenance and design improvement of the set of architectural design decisions: system cleanup, pattern detection and interpretation, cloning (or reusing) architectural knowledge, and integrating one set of decisions with other decision sets. Pattern detection of design decision sets (use case U12) is a relatively new field of research, so there is little theory in this matter to apply. However, the four visualization aspects are designed to collectively help explore a set of design decisions to discover new ideas and themes made apparent or visible through visualization. For example, in the graphical structure view, we can easily see the documented relationships between design decisions and a sense of grouping. Coherence and coupling can be determined with a single glance. In the decision structure and impact visualization aspects, we can identify isolated decisions that may be no longer relevant or find system components related to a decision that was just rendered obsolete (use case U7). Combined with tabular listing, a software architect can easily find a subset of decisions to clone or reuse (use cases U10 and U11) in new projects.

CHAPTER 5

ARCHITECTURAL DECISION TOOL DESIGN

In Chapter 3, I described the needs for a system-based approach for architectural design decisions, where I also integrated and outlined the challenges, requirements and use cases for a tool-based solution. Moreover, in Chapter 4, I describe other ways to support decision capture and visualization by proposing three capture approaches and four visualization aspects to improve the decision capture processes and decision exploration for software development organizations. The next logical step is to design and implement a software tool that would satisfy or support as much of those requirements and use cases in the tool-based solution as possible, using the proposed capture approaches and visualization aspects. The goal of the tool creation is to determine the feasibility of the requirements and use cases of the tool-based solution through the development of the system-based tool. As well, the tool provides the capability of gaining immediate feedback from users of the tool about the practicality of the proposed capture approaches and visualization aspects. In this chapter, I discuss how I design and implement an architectural design decision tool I name ADDEX (Architectural Design Decision EXploration).

5.1 Tool Design Overview

The ADDEX tool is a system-based tool that consists of four components. These components are actually four sub-tools using a common framework to collectively address decision capture and exploration. A general system structure overview of the ADDEX tool is shown in Figure 6. Three of these tools (i.e. the components) respectively address one of the three approaches to decision capture and store the captured decisions in a database using a common decision representation model and supporting data manipulation software framework. The fourth tool leverages the captured decisions to visualize and explore the different facets of the architectural knowledge found within those design decisions. All four tools are written in Java and can interface with an SQL relational database provided by the common framework. An overview of the ADDEX tool is also found in (L. Lee & Kruchten, 2008b).

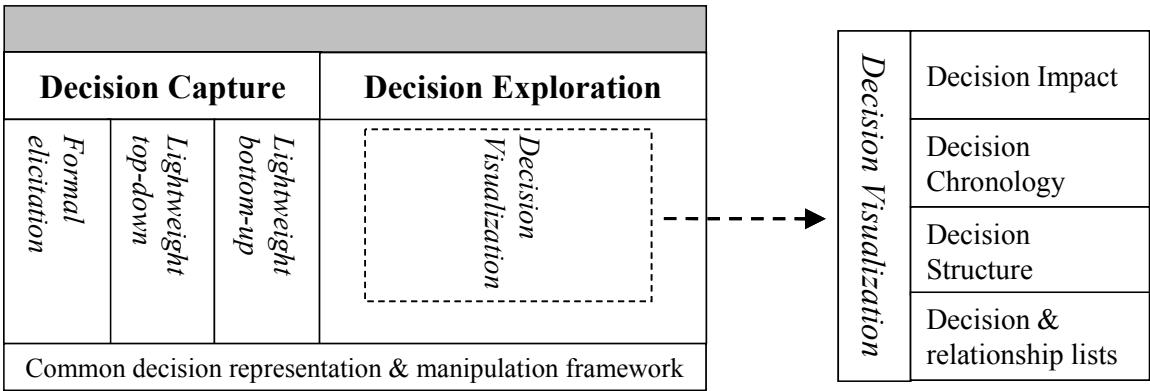


Figure 6: ADDEX system diagram. Four sub-tools make up the ADDEX tool and are tied together through a common framework for decision representation, storage, and manipulation. The visualization tool contains four distinct visualization aspects that can be used to explore architectural design decisions.

5.1.1 Decision Attributes

To represent the design decisions in all four tools, I used the ontological decision representation model, where decision details such as the description, rationale, scope, and state, and decision relationships are gathered using the tool. The selection and brief overview of this decision representation model is described previously in Section 2.5. For the two tools implementing the lightweight decision capture, the focus was on breaking down the capturing process to smaller steps, thus reducing the amount of immediate effort needed to capture architectural design decisions. In this case, I use a subset of the decision attributes required for the initial capturing event, and I provide additional support attributes so that the decision capturer can revisit the semi-documented decision at a later time to add in the necessary details. For example, in lightweight top-down capture, the required attributes during the initial capture event would be the epitome of the decision, who made the decision (or who documented it), and when was it documented. The additional requirements include a reference and an index to the source of the decision (in a document or meeting minutes, for example), as well as a casual “notes” section to document any additional information to informally remind the decision capturer about the decision at a later point in time. Table 20 summarizes the attributes used to capture design decisions in the three capture tools.

Table 20: Design decision attributes implemented in each of the three capture tools. Attributes denoted with a ‘*’ are additional attributes used to model the decisions for the implementations of the capturing approaches. Attributes in parentheses are implicitly implemented.

Formal elicitation	Lightweight top-down	Lightweight bottom-up
<ul style="list-style-type: none"> ▪ Epitome ▪ Rationale ▪ Scope ▪ State ▪ Categories ▪ Author ▪ Date/time ▪ (Publicity level) ▪ (Source) 	<ul style="list-style-type: none"> ▪ Epitome ▪ Author ▪ Date/time ▪ Source* (documents/media) ▪ Notes* ▪ (Publicity level) ▪ (Source) 	<ul style="list-style-type: none"> ▪ Epitome ▪ Tag source* (design artifact) ▪ Author

In the lightweight approaches, I am not implying that the reduced set of attributes are more important than other attributes found in the formal elicitation approach; however, I am stating that the subset of attributes are essential to the *particular lightweight step* being performed to capture decisions before it would be forgotten and lost. In other words, I am suggesting that upon completion of all the steps of the lightweight approaches would have not only *captured the same set* of decision attributes as in the formal elicitation example, but additional support attributes like traceability to decision sources, context, and relevance are captured as well.

For the scope and purpose of my research, the “source” attribute is intentionally folded into the rationale of the design decisions. I made an assumption that people can document the source of their decisions within the rationale. Since lightweight bottom-up capture documents decisions directly within the source, the “source” attribute is defined to be the same as the “tag source” attribute. Future work should make the source more explicit to better support traceability and intentionally tacit decisions. However, for the “publicity level” attribute, limited resources and scope dictate the implementation of personal and public design decisions, as it entails significant implementation resources to develop a supporting framework for user groups and security policies within the four components of the ADDEX tool. Publicity levels are implicitly defined in the formal elicitation tool and the lightweight top-down tool in that all the personalized decision flags in the lightweight top-down capture tool can be kept personal and private, while all formed decisions in all three decision capturing tools are set to an organization-wide level of disclosure. Since software artifacts are generally organization-wide in nature, there is little need

for publicity levels in lightweight bottom-up decision capturing. The context of decision set acquisition (see Section 6.2.2) ultimately limit the study of publicity levels, so no further attempts are made to modify the tool to support various levels of decision disclosure.

5.1.2 Users of the Tool

The ADDEX tool is designed to be used in a multi-user, collaborative environment. The targeted users for the tool are software architects, designers, developers, maintainers, and other stakeholders (refer to Table 14 in Section 3.2.1). It is assumed that the users are familiar and comfortable with computer technology. Each user is associated with a username and must be signed in to create and manipulate design decisions within the system. Currently, anyone can view the decisions, as I assume that all users accessing the system are authorized (e.g., the tools are operated within an organization's secured building and computer network). Created decisions and decision changes are associated with a user and the date and time of the change. The tools can be deployed in a distributed environment where users in various physical locations can simultaneously use the tools on their own machines. Alternatively, the tool can support multiple users sharing time on a single instance running on a computer. In this way, software organizations can customize the use and integration of the tools to their own specific needs.

5.1.3 Decision Storage and Retrieval

To store and retrieve captured design decisions, the ADDEX tool (with its four components) supports flat files, databases, and XML representation. The captured design decisions can be stored in a database located locally or remotely. To support decision storage into a relational database like MySQL, the general logical structure models the various entities in relations to a decision entity, as shown in Figure 7. If a database is not available, then the common framework can use flat files instead. Also, decisions, relationships, and other attributes can be imported and exported through external XML files. All decisions and relationships are assigned globally-unique identifiers to import only the decisions that do not exist in the system yet. (The underlying assumption in importing and exporting is that a group may collaborate with other groups on different projects or organizations, so they may want to import new decisions made by the other group. This means that decisions do not have to be system-unique, but globally unique. This also mitigates confusion when decisions from other projects appear unintentionally in

another project.) The use of XML for decision import and export satisfies the interoperability requirement (requirement V3).

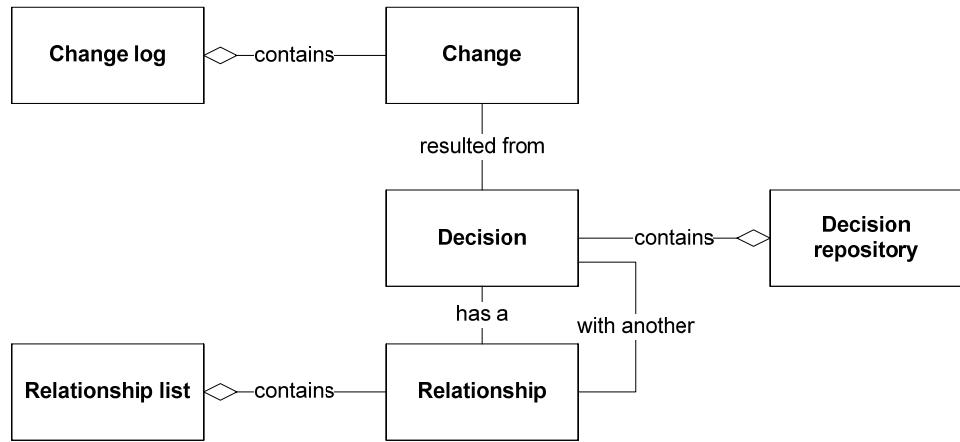


Figure 7: UML diagram of the common framework's basic decision representation structure

Support for incremental decision changes, like change history, is a fundamental part of the decision storage and retrieval framework. Each decision contains a change log, where each version of the decision is tracked and can be accessed. The underlying assumption is that the decisions and their previous versions are not deleted, but are rendered as rejected or obsolete. The common decision representation and manipulation framework has been designed in such a way that a tool using the framework does not need to know how the decisions are stored, retrieved, or modified. The next few sections describe how these tools are designed and implemented.

5.2 Decision Capture Tool Implementations

Of the four components that make up the ADDEX tool, three of them are tools to capture and manipulate design decisions (to fulfil use case U6). Each of these three tools can operate independently of each other, yet are able to share decision sets and resources through the common framework. The three tools can coexist on one computer or can exist on other computers in a distributed environment (when collaborating with other people—requirement R10); in both cases the three tools would not interfere, but help capture decisions in cases where the other tools are less capable. The following section describes how the three tools are implemented.

5.2.1 Formal Elicitation

The formal elicitation tool is designed to be used in highly technical architecture discussion sessions where decisions are created or modified in batches in response to a technical discussion. Capturing descriptive design decisions in a methodical, structured form is possible and conducive in a technical environment. Decisions are captured and represented formally (requirements R14 and R15) using the decision representation model described in Section 2.5. Structured knowledge helps maintain knowledge consistency and assists in identifying conflicts after the decisions are captured (requirements R2 and R3). The tool's uncluttered interface focuses on the utility of capturing design decisions and allows users to browse and modify the collection of decisions for a project. Users can select a particular project to browse its collected decisions, but they must log in to the system in order to create, edit, or remove decisions. The support for multiple users addresses the groupware and collaboration capability (requirement R10), but the tool can be just as useful as a personal decision capture tool (requirement R16). The tool uses an SQL database to store and retrieve the captured decisions. Decisions are never deleted from the system, but rendered obsolete to increase system traceability and maintain a temporal flow to the capturing process (for use cases U1 and U5, which deal with decision chronology). I would like to highlight two scenarios for this capture tool—decision browsing and decision elicitation/maintenance.

In decision browsing, a user is shown a list of captured decisions. Each decision can be selected for viewing where a new dialog would show the decision's attributes (such as the description, rationale, scope, state, and change history). This view inherently supports decision querying (requirement R6). Other details can be captured, including decision relationships. The user does not have to be logged in to browse decisions and their details. In decision elicitation/maintenance, a user is required to log in and will then be shown a list of captured decisions similar to decision browsing. 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 8 depicts the formal elicitation tool while a user is saving a decision. Editing decisions is similar to browsing the details of a decision, but all the fields can be edited and saving changes require another change log entry. Thus, a history of creation and edits are tracked for

maintenance and traceability. Decision editing implies the ability to gradually form decisions by updating the decision with additional or more appropriate information (requirement R11). The documented flow of changes is useful when studying the decision chronology and changes (use cases U1, U3, and U5).

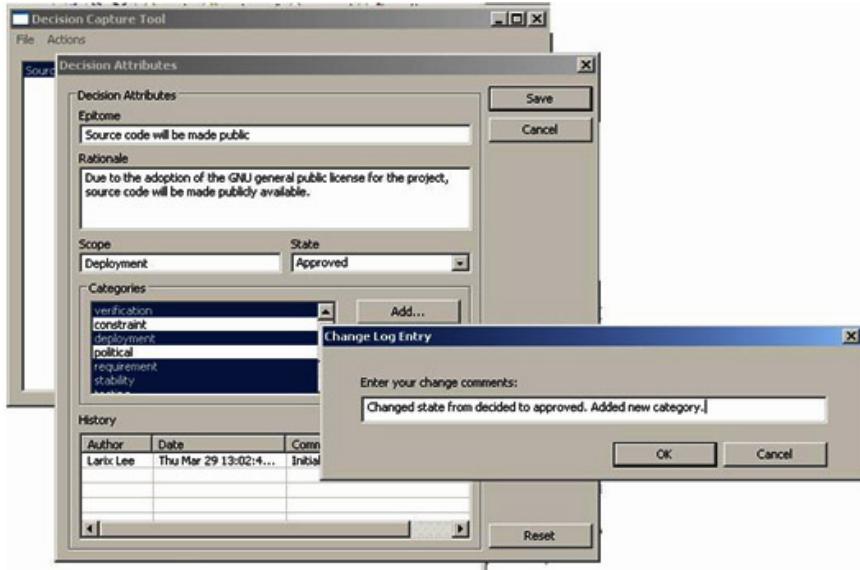


Figure 8: Screenshot of the formal elicitation tool. This figure depicts an open “decision details” dialog during an edit. The user has just hit ‘save’ and is prompted to enter a change comment. Figure from (L. Lee & Kruchten, 2008a) with permission from IEEE.

5.2.2 Lightweight Top-down Capture

The top-down capture tool implements the flag-filter-form method. Nicknamed “DecisionStickies”, the tool attempts to model after sticky-notes (or PostIt™ notes). This tool, shown in Figure 9, is based on the way someone can write on and use sticky-notes as bookmarks for later information retrieval. After applying this usage metaphor to a decision capturing process, the result is the creation of a capturing tool that has a familiar interface and supports lightweight, low-impact capture (requirement R1). The tool uses the same infrastructure to store and retrieve users, projects, and decisions, as in the formal elicitation process. However, the top-down capture tool focuses on the personalized decision capture (requirement R16). It also documents and describes decisions gradually and more freely (requirements R11 and R14). The differences are mainly the addition of a smaller data structure known as a decision reference, the way that data structure is stored, the absence of decision relationships, and the minimalist,

sticky-note user interface. The unobtrusive and familiar interface would help promote continual decision capture (requirements R17) because it can be better integrated within the software development process.

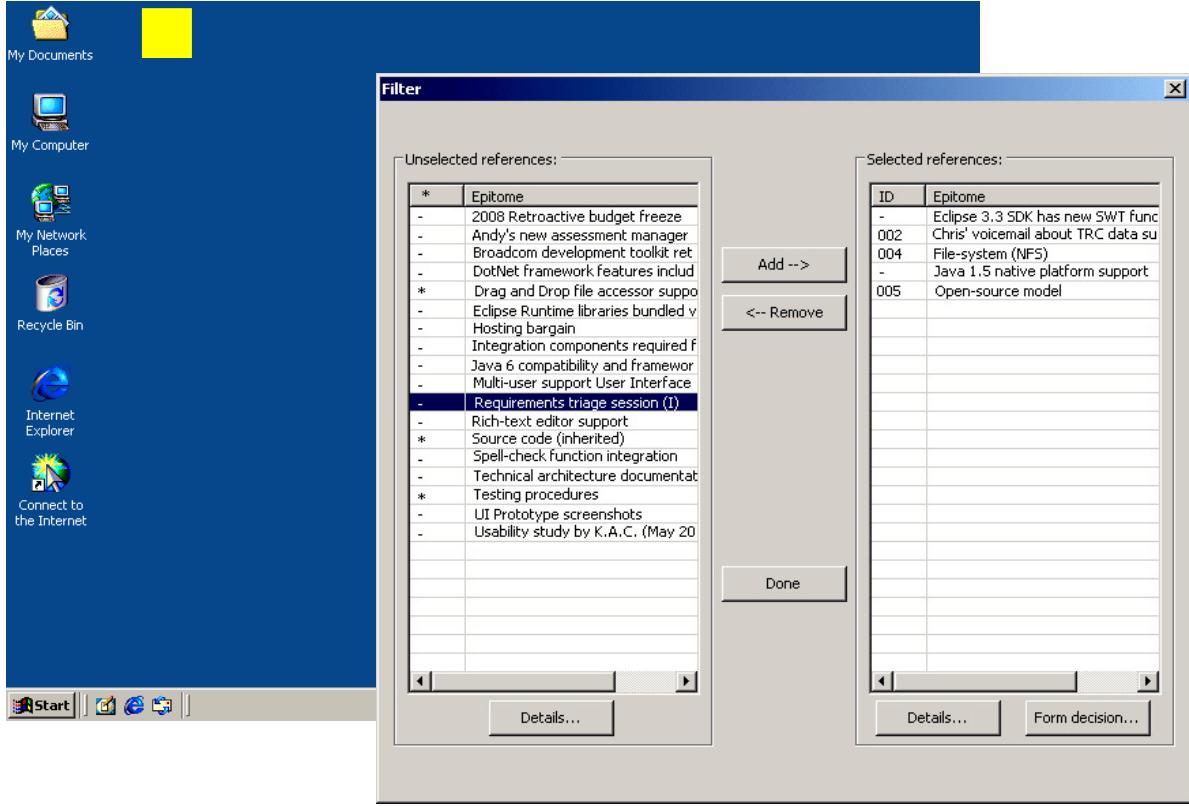


Figure 9: Screenshot of the top-down capture tool. Also known as “DecisionStickies”, the tool’s main interface is the yellow square box near the upper-left corner of the figure. In this figure, the tool is running on a typical computer desktop and the tool’s “filter” dialog is displayed. The left column of the filter dialog shows captured decisions references. The right column shows the decisions that are selected (deemed relevant) and can be formed.

The following is an example of how this tool functions. Once a user logs in and selects a project, the tool starts up with a little yellow square (the sticky-note) on the desktop, similar to a real sticky-note pad on an actual desk. A user can flag a decision reference by dragging-and-dropping a file or an e-mail onto the tool. The user is shown a dialog box containing a few text boxes for the user to enter some quick information about the decision reference. A few fields, such as the location of the document and the date are pre-filled so the user only needs to enter a decision title (i.e. the epitome) and the description of what the decision reference alludes to. Once done, the user saves the decision reference and can continue with whatever the user was working on.

previously. Decision references not in an electronic format can be added manually using the yellow sticky-notes square.

Over time, many decision references are captured for the project, and periodic sifting is required. The user can right-click on the sticky-note and filter the decisions. To filter decisions, the user is shown a screen with two lists of decision references: “available” and “selected”. Decisions references that are considered relevant are distinguished by placing them onto the “selected” list. The remaining irrelevant decision references are left in the “available” list for future browsing. Decision references can be moved between the two lists and the user can form the decisions similar to the formal capture tool. Decision flagging and filtering provide the ability to capture content specific to the needs of the designers and stakeholders (requirement R12). Irrelevant content would not be further documented. Decision forming is similar to the decision elicitation step in the formal-elicitation tool described previously with the exception that some fields, like the epitome and the author, are pre-filled with information found in the decision reference.

In the bigger picture, when other software designers and developers document their design decisions, a distributed collection of design decisions emerges. Gathering all the decisions together into a central repository or database establishes a corporate design knowledge base on the decisions and the background information on the database. I have already implemented support for this through a central SQL database. What keeps captured decisions personal is the support for varying levels of disclosure provided by the decision representation model. However, due to the limitations of time and scope, publicity levels are implicitly defined. Flagged and filtered decisions are personal and are stored in a private user-space, but formed decisions are organization-wide and made public. Any concern over organization-wide adoption to attain critical mass for tool usability or adoption can be assuaged because a primary goal of the tool is to capture decisions in a personal way (i.e., “memory-aid”). Groupware and collaboration support would then be an asset.

5.2.3 Lightweight Bottom-up Capture

Because the lightweight bottom up approach takes gradual decision capturing and formalization (requirements R1, R11, and R15) from the perspective of a software programmer, tester, or maintainer, it follows that a capture tool of this approach should be integrated into the tools of the developers. Tool integration improves adoptability and continued use (requirements R17 and V7). For the bottom-up capture tool, I focus on capturing decisions stored in software code, but bottom-up capturing can work with other software design artifacts like UML diagrams and technical architecture specifications. The idea is to establish a close-proximity to the low-level architectural design (requirement V12). When a user coding in a project encounters an architectural issue, the user would first consult with his or her peers and/or the software architect. The user would then make an architectural decision and the user would modify the code to reflect the design changes. Then, the user would tag the affected part of the source code with a decision comment or tag, and carry on with the work. In the meantime, the Eclipse Plug-in finds the newly added tag and displays it in a list within a view. As decision tags are intentionally created to quickly document a specific concern in the software design artifact, the tags are highly relevant to the software designer and other stakeholders (requirement R16). To support the collaborative software design and development environment, the tool may update other programmers working on the same project files, notifying them of the new decision via the interface (requirement R10).

As many developers have adopted the Eclipse Integrated Development Environment¹ (IDE) to be their programming environment, I 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. This tool is shown in Figure 10. The tool’s purpose is simple: parse through all the code in the project’s source files, identify all decision tags and comments, and display them all in a “view” within Eclipse. Though not currently implemented yet, the tool would form decision entities (requirement R15) in a way similar to the formal capture tool. The decision-forming interface is also similar to the formal elicitation tool. To form the decision, the user would right-click on the decision in the code or in

¹ Eclipse Open Development Platform. <http://www.eclipse.org>

the view and select “form”. This forming step can be performed during code check-in, but it is better to integrate with the code-review part of the development process, when the user can discuss the issues with other developers and share the decision knowledge at the same time.

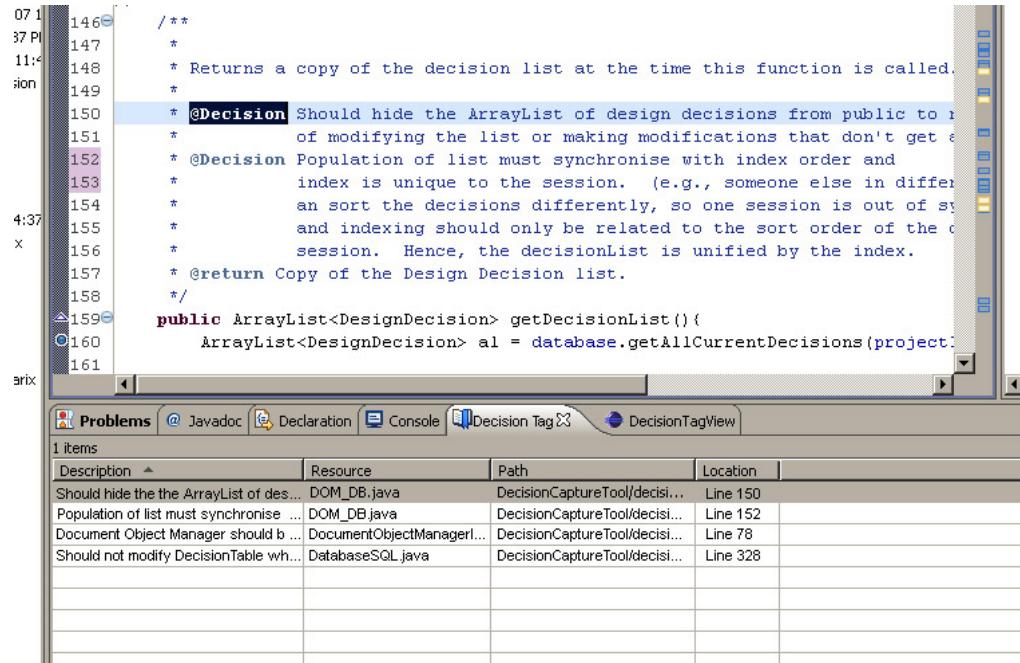


Figure 10: Screenshot of the bottom-up capture tool. The tool is implemented as an Eclipse plug-in. Decision tags are listed in the “decision tag” Eclipse view, where decisions could be formed using this view. Selecting a decision tag brings up the particular file and line in the source code where the tag is stored. Figure from (L. Lee & Kruchten, 2008a) with permission from IEEE.

5.3 Decision Visualization Tool Implementation

The fourth component of ADDEX is the tool that visualizes architectural design decisions (requirement R8). This tool visualizes software architectural design decisions separately from the software architecture in which the decisions reference. The reason behind this is to look at the decision view of design decisions as a knowledge repository where people can gather information about a design in a central location. The purpose of this tool is to facilitate decision browsing, editing, manipulation, and exploration without introducing significant complexity (requirements R9 and R13). My selected decision model has a higher capability of visual abstraction (requirement V9) for decision exploration than other decision models (see Section

2.4) due to the amount of associability provided by the explicit support for decision relationships.

The decision visualization tool is based on the same common decision representation and manipulation framework as with the capture tools. In addition to visualization, the tool is interactive in that a user can create, modify, and remove (make obsolete) both the decision and its interrelationships while visualizing the information. The tool utilizes the Prefuse visualization framework (Heer *et al.*, 2005) for the visual representation of design decisions. The Prefuse framework allows rapid development of visualization tools by providing a base structure for visualization, graphical support, automatic layouts (requirement V13) and visual interactivity (requirement V5). Rendering scalability (requirement V1) is mainly handled by Prefuse. A user visualizes the decisions in several different aspects to support decision perusal and exploration.

The tool has four main views (requirements R7 and V8) for decision visualization and information display. The first is a simple tabular list of decisions and their relationships, while another view visualizes the decisions using decision-graphs to display the decision structures and relationships. The tool can also visualize the decisions in a chronological order and the fourth view displays decisions from an impact perspective.

5.3.1 Decision / Relationship Lists

This view is the most common in the decision tools. The formal elicitation tool and, to a lesser degree, the two lightweight capture tools, use this visualization aspect for its decision representation abstraction. The decision / relationship list simply lists the design decisions in a table, showing a selection or all the attributes of a design decision. Decision relationships are also listed in another table that references the decision list. A screenshot is depicted in Figure 11. The purpose of this view is to supply a quick and effective way to browse and retrieve information (requirement R5) from design decisions. The textual representation of the decisions facilitates decision querying and filtering (requirements R6, V10, and V11) as well as simple decision entry. However, it is difficult to trace decision relationships and quickly assess decision properties when the decision set becomes large. Information scalability (requirements R4 and V2) becomes dependent on how well queries and filtering are formed and executed.

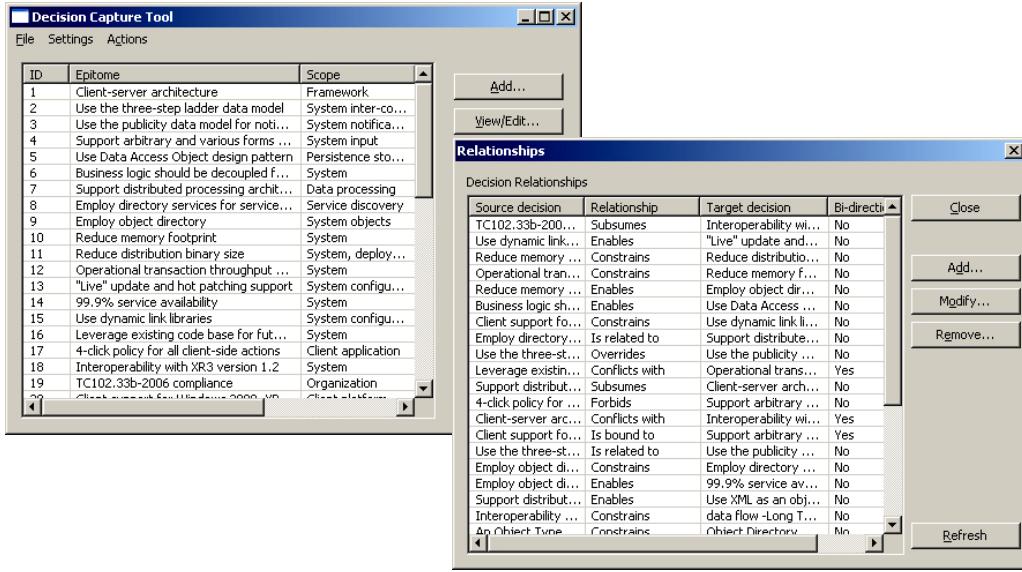


Figure 11: Decision and relationship lists for a set of decisions. The lists show the current set of design decisions and their relationships. Users can add, remove, and peruse the captured decisions and their relationships by double-clicking or selecting a row in the list.

5.3.2 Decision Structure Visualization

With large decision sets, an effective way to sort and analyze decision information is to abstract and represent the decisions graphically (requirement V9). In this view, we visualize decision graphs, in which decisions are represented as nodes and the relationships are the edges. Figure 12 depicts a decision graph that represents the decisions and their relationships. Decisions and relationships can be created, selected, viewed, modified, and removed from this view. The advantages of graph visualization are apparent: an observer can see relationships and their associated decisions more quickly than from a list. Moreover, the observer can also assess the level of knowledge or design completeness by looking at the number of isolated nodes. A well-documented project would have many interconnected decisions. For example, a large proportion of isolated decisions could govern mutually exclusive feature sets, but there is likely a set of decisions that ties all these features together into the software system. The missing relationships draw attention to the missing set of decisions. Decision relationships promote design cohesion and solidity during software design, so it is beneficial to be able to view the relationships easily.

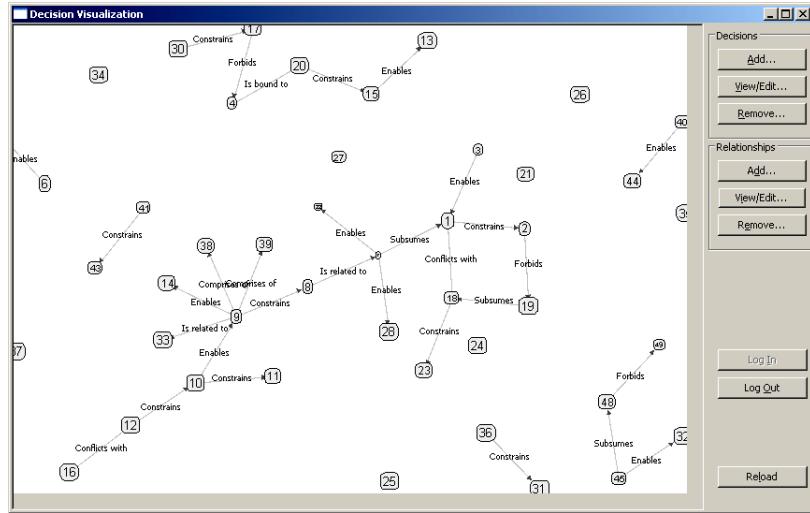


Figure 12: Decision structure view of a set of decisions. This screenshot shows the visualization of design decisions and their relationships as a directed graph. The nodes represent the decisions and the directed edges represent the decision relationship to another decision. The size of the node denotes the decision state – for example, larger nodes represent decided or approved decisions while smaller nodes represent ideas or tentative decisions.

Besides the view's graphical visualization, there is a high degree of interactivity (requirement V5) to communicate information. Using a force-directed layout (requirement V13) for the visualization of the decision graph, the tool represents decisions of a less mature state as being physically lighter in the layout model and visually smaller than more mature decisions. I intend that the maturity of a design could be visually assessed from the number of small or large nodes in the graph. The capability to assess design maturity from the size of nodes implements an instance of the pattern detection and interpretation use case (use case U12). When the user interacts with a decision node or a cluster of nodes, the user could assess the maturity from how quickly the decision can be moved around the screen. For example, more mature decisions have more “weight” (they are inset into the design and have significant inertia), so the decision nodes behave like heavy objects in the view.

Depending on the zoom level, the decision nodes can show more or less information about the decision. Known as “semantic zooming”, this strategy avoids overwhelming users when they visualize large decision sets (requirements R4 and R5) and helps with information scalability (requirement V2). When a user zooms towards a decision, the decision’s properties will appear

inside the node. When a user zooms away, decision information gets hidden. Viewing the decision or relationship details can also take place without zooming simply by selecting a decision. Figure 13 illustrates this semantic zooming feature.

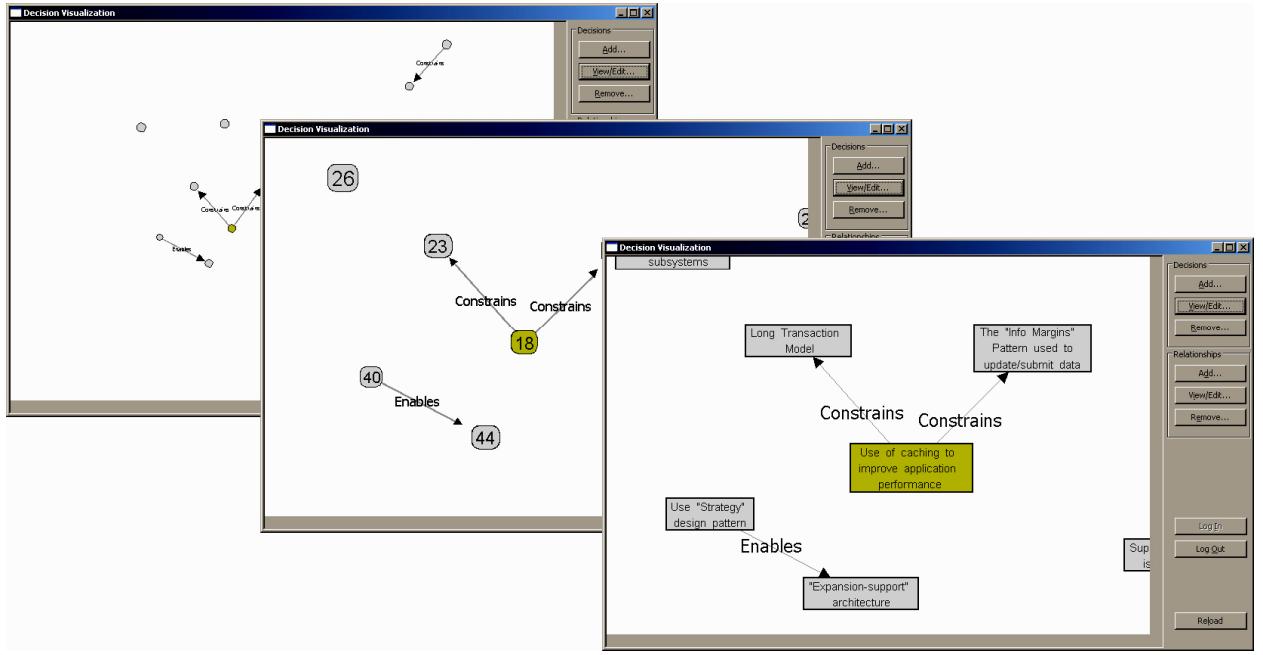


Figure 13: Semantic zooming in the decision structure view. When the user zooms in or out, the amount of decision information being shown on the screen will increase or decrease respectively. In this figure, the centre screenshot depicts the default zoom-level. The upper-left screenshot depicts what the user sees when the user zooms out (decision identifiers are hidden). The lower-right screenshot depicts what the user sees when the user zooms in (the decision epitome is shown in lieu of the decision identifier). The yellow node highlighted is the selected decision. The decision epitomes have been modified in the screenshots for decision set confidentiality.

5.3.3 Decision Chronology Visualization

The tool supports a time-based view of design decisions to show the evolution of design decisions and gradual formalization (requirement R11) and provide the ability to quickly determine created or changed decisions during a specified time interval (use cases U1 and U5). This view is shown in Figure 14. A user can select a subset of these decisions to view in more detail (requirements R4 and R5, use case U4), such as the decisions within a cluster, and can create, view, or modify decisions (requirement R13).

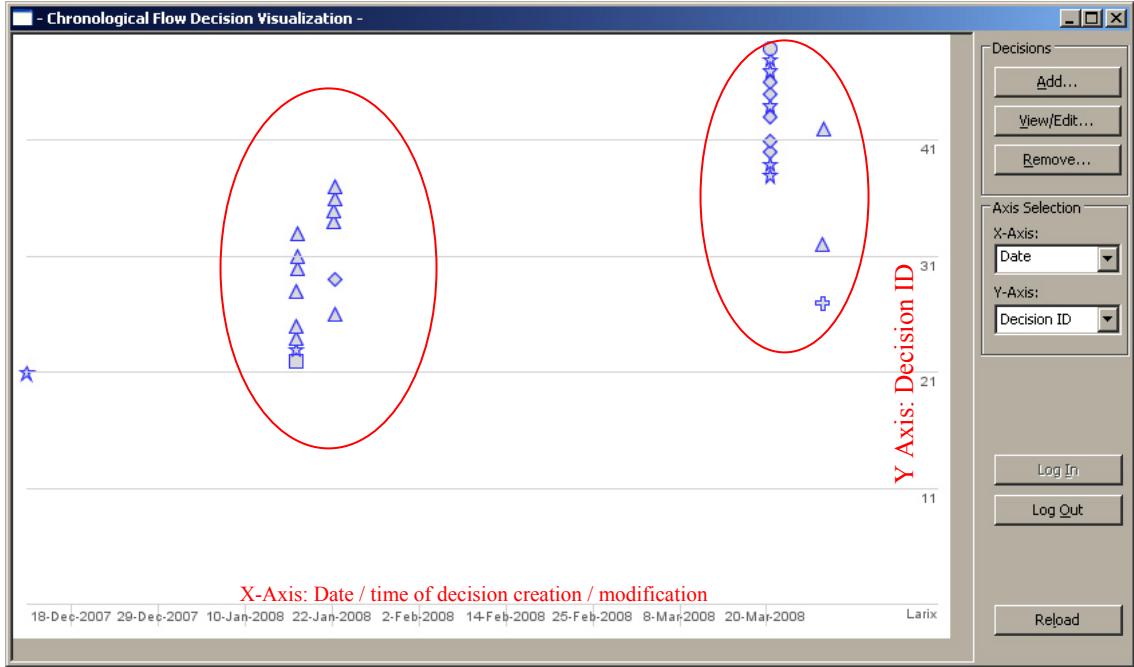


Figure 14: Chronological view of a set of design decisions. This example shows two decision creation and management periods (highlighted with red circles) over a three-month interval. Decision state is denoted by the shape: circles are ideas, triangles are tentative, squares are decided, stars are approved, crosses are challenged.

This view initially displays all the decisions created and modified during the project in a timeline, with the date on the x-axis and a user-selectable field for the y-axis. Decisions that are closely spaced denote a decision capture or management session. A user can quickly identify the state of a decision by its shape in the view (use case U12).

A particular area of interest is in the user-selectable y-axis, which supports a light querying implementation (requirements R6 and V10). The tool currently allows categorization of the y-axis by decision ID or decision author. If the decision ID is used for the y-axis, one can view decision changes in a global perspective (because the decision ID is implemented as an increasing number). If the author is used for the y-axis, we can determine which decision-makers are most active and which changes they have made. Categorizing by author includes the ability to find both subversive and critical stakeholders who can potentially damage the system if they change their minds (use cases U8 and U9). Figure 15 depicts an example of categorizing by author. By customizing the user-selectable y-axis with other criteria types (requirement V4), the tool enables people to find and use hidden knowledge within design decisions (use cases U2,

U12) in allowing people to make associations with various criteria to find patterns not easily visible.

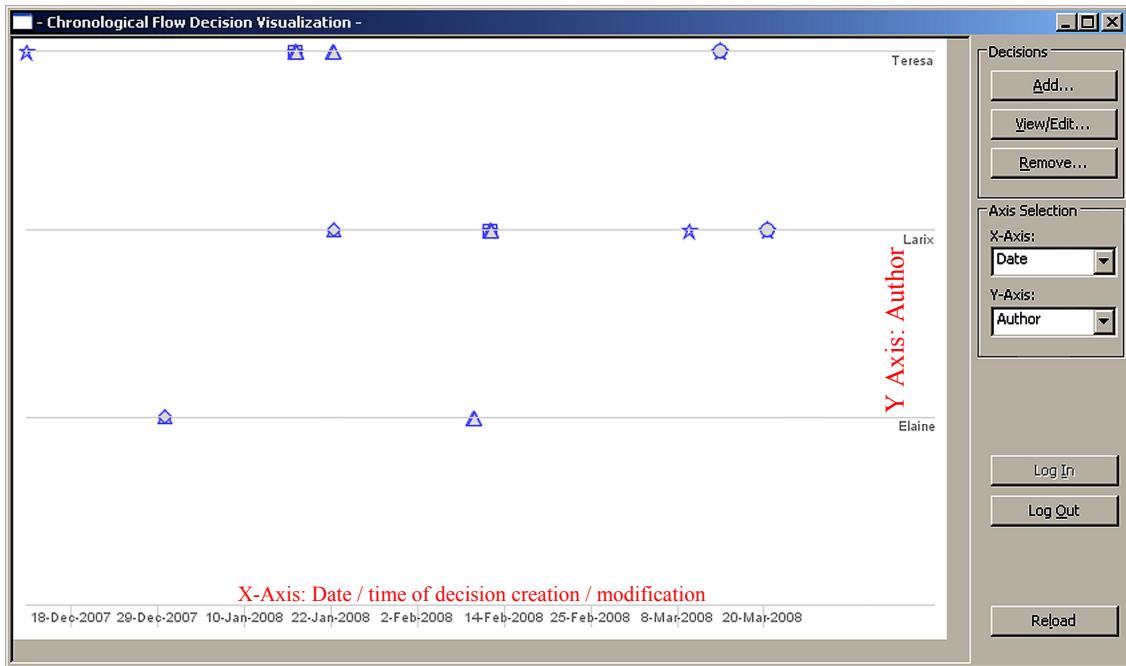


Figure 15: Chronological view of a set of design decisions: Categorized by author. In this view, decisions are grouped by author and are showing which stakeholders or designers are most active during a three-month period, and finding the stakeholders' interests and foci are possible by looking for and viewing clusters of decisions.

5.3.4 Decision Impact Visualization

The fourth view of design decisions that this tool supports is decision impact. Shown in Figure 16, this view provides a visualization of decisions that can be potentially impacted by a change of a decision. Though the decision structure visualization supports visualization of decision relationships, there are related decisions that are associated by attributes, such as author, scope, and categories. The tool provides an entry-point into the large matrix of potential impact-relationships by visualizing it to support decision impact analysis (use case U3). The impact-relationships also promote knowledge consistency (requirement R3) through concern-based design reviews (use case U2) because it enables people to find other decisions of concern through common attributes to identify and resolve conflicts (requirement R2). An example impacting concern would be, “What are the other approved decisions related to authentication using remote database deployment if Windows-authentication will be used?”

In this visualization, decisions are laid out automatically (requirement V13) using a radial layout, where all other decisions surround the selected centre decision. Like the other visualization components, the decision impact visualization is also interactive (requirement V5). Selecting a different decision brings that decision into the centre and all other decisions surround it. Resting a mouse cursor on a decision would highlight neighbouring decisions associated with an impact-relationship. The impact relationships can be filtered (requirement V11) according to different criteria, such as category, scope, or relationship. Currently, the tool links decisions that share a common criteria value with an impact-relationship, though the tool can be modified to support customized filtering and queries involving different criteria values, ranges, and thresholds (requirements R6, V4, V10 and V12).

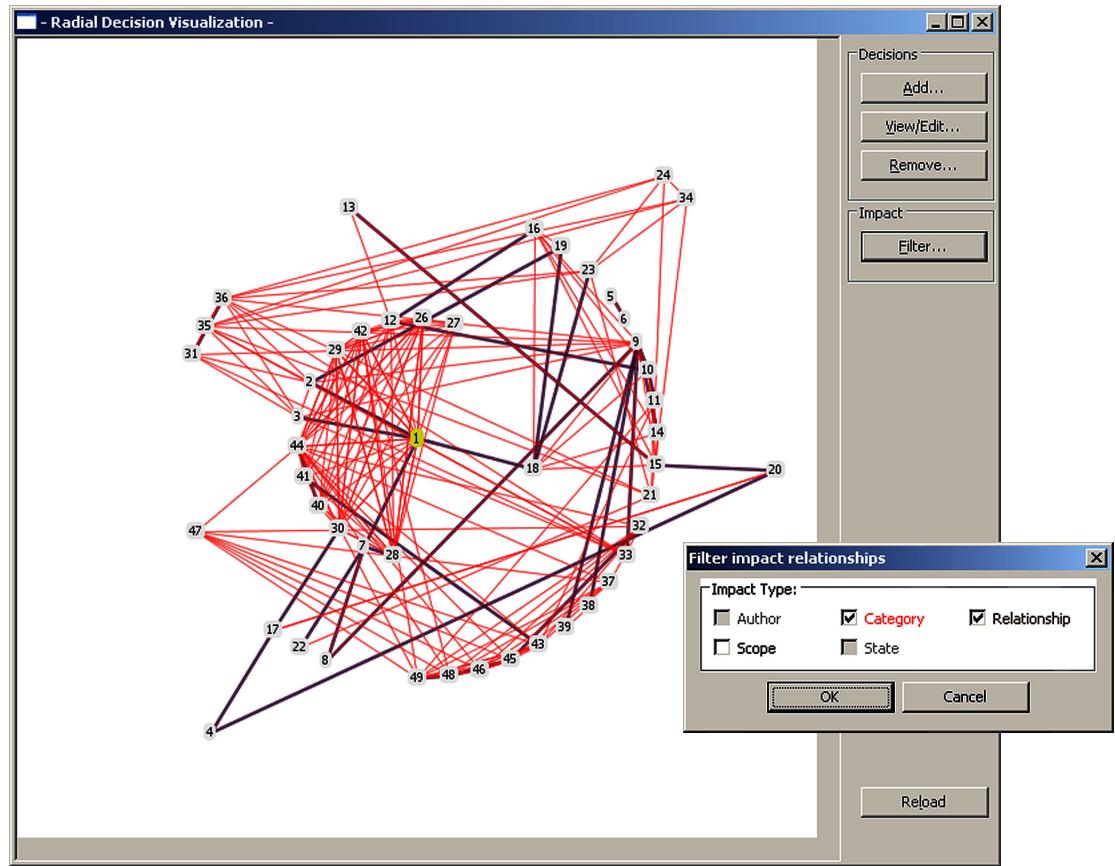


Figure 16: Decision impact view of design decisions. The nodes represent design decisions while the coloured lines represent the impact-relationships between them. Thick edges are the decision relationships and thin edges are impact-relationships (i.e. “category” in this case). The highlighted centre node is the decision in concern. Immediate (and outer) neighbours to this node are decisions that are directly (and indirectly) impacted by it.

5.4 Comparison with Other Current Decision Tools

There are a number of tools created recently for attempting to capture, represent, and utilize design decisions; some are from the design rationale community and some are from the architecture community. Many of these tools were created for the purpose of demonstrating the knowledge or decision representation model, so decision capture in those tools is considered to be a means to an end. In other words, decision capture is not often explicitly addressed. Visualization support in these tools is rare and those that do focus more on the decision set model and less on the use of the visualization for decision exploration. I will briefly look at a few of these tools in the context of decision capture and exploration.

A closely-related tool is PAKME (Process-centric Architecture Knowledge Management Environment), which is a web-based design decision tool that focuses on general architectural knowledge capture and management of scenarios, patterns, design options, and decisions for the software architecture process (Babar *et al.*, 2005, Babar *et al.*, 2006). PAKME's decision capture approach focuses on capturing decisions throughout the entire development, where architectural knowledge (including decisions) can be added and updated at any point in time. PAKME addresses two strategies to capture and present knowledge: the first is elicitation by individuals or teams and the second is knowledge creation throughout the software development process – reminiscent of Nonaka's “Knowledge Creating Company” (Nonaka, 1991). The former strategy can be represented by the formal elicitation capture approach, while the latter strategy suggests the combination of lightweight top-down and bottom-up capture approaches integrated in the same tool. In terms of visualization, the web-based system is heavily textual, relying on tabular listing for decision exploration. PAKME takes full advantage of the query support inherent to the textual tabular listing and decisions can be easily retrieved, parsed, and edited. The tool favours architectural knowledge creation, browsing, and management.

Another architectural knowledge capturing, representation, and management tool that is closely related is the AD_{kwik} tool (Schuster, 2007). AD_{kwik} was created by Schuster for IBM Research as part of her doctoral thesis and the tool ties together the ideas of design decision dependency management, decision workflow/process support, design knowledge repository, and design collaboration. AD_{kwik}'s decision-making process involves three steps that could be performed in

parallel on a set of decisions: decision identification, decision-making, and decision enforcement. The first two steps echo the lightweight top-down approach as it captures decisions, while the third step addresses decision updating, management and maintenance. Like top-down capture, AD_{kwik} draws on many mediums to capture and store decisions (Wikis, files, e-mail, and message boards) and brings them together into a common environment. The environment also supports formal elicitation by performing all three steps at the same time. Decision exploration support is provided through the Web 2.0-based interface, with structured hierarchical lists and guided interfaces (“next steps”), which promote the sense of continuity and design flow. Like PAKME, decision visualization in AD_{kwik} is highly organized tabular listing and text-based.

There are other recently developed decision capture tools. The SEURAT tool is an Eclipse development environment plug-in utility that captures and displays design rationale while developing and maintaining software code (Burge & Brown, 2006). The SEURAT tool focuses on the uses of design rationale so the tool only briefly addresses rationale capture. Its tight integration with the Eclipse environment allows design rationale to be captured; however, the goal of SEURAT is to assist in software maintenance, focusing less explicitly on software architecture. Design decision exploration is through a hierarchical tabular listing within an Eclipse view where its structure closely resembles the decision representation model it uses. Another rationale-based tool, Sysiphus, is a toolset that assists in the capture of various system models for system various development activities (Bruegge *et al.*, 2006). It supports rationale-based design decisions and links them with system models, use cases, requirements and test cases. Traceability is an important feature the tool addresses. It uses interactive, focussed graphs to visualize and explore the complicated traceability relationships (the edges) between actors, use cases, requirements, and test cases (the nodes). A decision is represented in the graph as a collection of visualized design rationale attributes like issues, options, and criteria.

There is also the Compendium tool (Selvin *et al.*, 2001), which documents the flow of knowledge and design rationale during interactive team meetings. The Compendium tool derives from the IBIS-based approach proposed by (Sierhuis & Selvin, 1996). Compendium is a general knowledge and decision capture tool, but there are concerns that it does not apply well to

architectural design decisions (Jansen & Bosch, 2004), where it lacks in describing first-class architectural concepts such as the various types of interaction between components (e.g., inheritance, data flow, or aggregation). This could be attributed to Compendium's focus on argumentation modelling. Nevertheless, it does model decisions and is a reasonable decision capturing tool. Compendium models the flow of decisions graphically, where each design decision is represented as a node and the subsequent refinement or addition of related decisions would result in appending those decisions after the initial decision. The influences and dependencies of these decisions are represented as edges in the graph. The graphical nature of Compendium makes design “replay” and traversal easier. However, Compendium, Sysiphus and SEURAT capture design rationale in a formal elicitation approach in that decisions are elicited directly to a formal model.

The tool for the Archium approach is an architectural design decision tool which primarily focuses on how software architecture can be represented as a set of design decisions; focussing on decisions can be traced to the requirements and to the architectural components of a software architecture (Jansen *et al.*, 2006). Archium regards design decisions as a “change function” with a single parameter (Jansen & Bosch, 2005), where decisions are linked to the architectural components and connectors, and decision dependencies are modelled. The focus of this tool is to demonstrate the Archium approach and the structure of the design decisions. Architectural components and requirements are visualized graphically as distinct nodes connected together through change functions, and the general graph constitutes a decision. Another tool, the ADDSS (Architecture Design Decision Support System) tool, is a web-based tool to capture and document architectural design decisions for immediate browsing (Capilla *et al.*, 2006). Like the other web tools, it suffers from the limitations of the web interface. The tool lists the system requirements, the decisions, and the requirements it addresses in a tabular list, although it supports the display of user-uploaded picture files to represent architectural products of arbitrary format. Although the current decision capture approach can be classified as formal elicitation, another version is being developed that will integrate with software tools used by architects (Capilla *et al.*, 2007), suggesting a bottom-up capture approach. The new version would also involve better decision visualization.

IBM Research also developed an Eclipse-based tool, called the Architect’s Workbench, that tries to balance the architects’ formalism and freedom of expression to structure and organize architectural knowledge “into sufficiently formal work products” (Abrams *et al.*, 2006). Architect’s Workbench uses wizards (step by step query processes) for many complex tasks to create or document design knowledge, similar to the lightweight approaches in that the wizards break tasks down into multiple steps. This tool supports various forms of knowledge visualization for exploration. The increased freedom of expression used in architectural knowledge capture resulted in the proportional increase freedom and capability of knowledge exploration. Knowledge can be documented and visualized in structured tabular lists and its unstructured-form version (simple text fields). A free-form graphical area can display knowledge and relationships using essentially whatever graph syntax the knowledge capturers desired. The freedom of expression lends a level of flexibility towards decision exploration by not hindering customized styles and notations.

5.5 Meeting the Requirements

After designing and creating the ADDEX tool, I should check the tool’s implementation against the requirements and guidelines as described in the earlier chapters (refer to Table 17) to verify whether it is possible to build the tool-based solution as described. The ADDEX tool attempts to accomplish as much of the requirements as possible. A use case comparison is shown previously in Table 19 and a summary of a requirements comparison is shown in Table 21.

Table 21: Requirements traceability matrix for ADDEX

Requirements (Source)	ADDEX Tool Components						
	Decision Capture			Decision Exploration			
	<i>Formal elicit'n</i>	<i>Lgtwgt. top-dwn</i>	<i>Lgtwgt. bttm.-up</i>	<i>Tabular listing</i>	<i>Graph'l structr.</i>	<i>Chrnlg</i>	<i>Impact</i>
R1: Capture with minimal overhead	---	Yes	Yes	---	---	---	---
R2: Resolve conflicts	Support	---	---	---	Support	---	Support
R3: Knowledge consistency	Support	---	---	---	Support	---	Support
R4: Retrieval strategies to manage large datasets	---	---	---	Yes	Yes	Yes	Yes
R5: Retrieve info. without navigating through all data	---	---	---	Yes	Yes	Yes	Yes
R6: Support querying	---	---	---	Frmewk	Frmewk	Partial	Partial
R7: Multi-perspective	Yes			Yes			
R8: Visual representation	---	---	---	Yes			
R9: Complexity control	Yes			Yes			
R10: Groupware/ Collaboration	Yes						
R11: Gradual decision formalization	Implicit	Yes	Yes	---	---	View	---
R12: Stakeholder-specific content	---	Yes	Yes	---	---	---	---
R13: Easy content manipulation	---	---	---	Yes	Yes	Yes	Yes
R14: Descriptive in nature	---	Yes	Yes	---	---	---	---
R15: Knowledge codification	Yes	Yes	Yes	---	---	---	---
R16: Knowledge personalization	---	Yes	Yes	---	---	---	---
R17: Sticky in nature		Yes	Yes	---	---	---	---

The two capturing tools implementing the lightweight capturing approaches address the specific need to capture with minimal overhead by breaking down the capture process into smaller steps. Guided by a decision representation model, consistency checks and conflict identification could be performed during the formal elicitation decision capture and through the graphical structure and impact analysis visualization aspects. However, the consistency checking and conflict awareness are limited to manual identification. In the next release, more automatic conflict identification can be performed through the comparison and cross-referencing of keywords and other decision attributes so that conflicts could be identified upon decision entry. To effectively

handle situations involving large decision datasets, visualization techniques were used alongside the strategies provided by using a relational database to retrieval and navigate through large amounts of information. For example, the graphical structure visualization aspect uses semantic zooming to reduce or increase the amount of information shown to the user. Decisions of less interest are culled from the user's view, yet more decision details would be displayed for the decisions currently in view. Other visualization aspects also employ filtering techniques and conceptual simplification through visual cues and information encapsulation.

For the ADDEX tool in general, the framework for decision querying is in place. All decision creation, retrieval and manipulation functions are performed using SQL queries. Unfortunately, due to limitations of time and resources, certain features have priority and I am not able to fully implement user-side querying; however, I am able to implement a fixed-query support in the form of selective filtering in the decision chronology and impact aspects. Query support would be especially useful in the tabular listing and graphical structure aspects to help find and reduce a set of decisions to explore. In hindsight and after acquiring feedback from industry, query support is a component that should have received a higher implementation priority.

The ADDEX tool applies well to the decision view requirements of software architecture. The ADDEX tool implements multiple perspectives in both the capture (formal meetings, early-stage design, and development/maintenance) and the exploration (the four visualization aspects). Visual representation is apparent in the four visualization aspects and complexity control is covered through the customized capturing processes and the visualization techniques to handle large amounts of data. As the tool is designed for a collaborative and distributed environment, groupware support is a fundamental part of the ADDEX tool. Gradual decision formalization is achieved through the customized, decision capture approaches that break down the capture into smaller steps, which can be performed in different sessions. This chronological flow is also visualized in the chronology visualization aspect.

The final set of requirements mentioned include whether the captured content is meaningful to the stakeholders and could be created and manipulated easily. The interactive visualization tool allows decisions to be created and modified via a couple of mouse clicks while the three

capturing tools allow stakeholders to capture only what is needed. Moreover, the lightweight capture approaches support informal annotation, so software architects and designers can be more expressive and descriptive during decision capture. As a result, decisions are more personalized, yet are also more formal and structured, because the captured decisions are structured using a decision representation model. Since a goal of the tool-based solution is to encourage decision capturing and promote decision exploration through visualization, the ADDEX tool should be used frequently by the architects, designers, developers, and maintainers. However, confirming this requirement requires a long-term study (6 months or more) on how a software organization would use this tool for their software projects, and whether it could meet the “sticky-in-nature” requirement that could not be tested and traced (refer to Table 21) during the scope of my research. Long-term usability study is a direction I should investigate in future work.

The implemented ADDEX tool must also support the visualization tool requirements as well. Table 22 compares the ADDEX visualization components to the visualization requirements. The choice to use the Prefuse visualization toolkit makes the visualization tool requirements easier to attain because it already encompasses many desired attributes and requirements. About half of the quality attributes are provided or inherently supported by the visualization toolkit. For example, rendering scalability (V1) is handled by the Prefuse rendering engine for the most part. Information scalability (V2) is supported with the tool’s internal query-support (V10) and the use of attributes to structure and build a graph. Further improvements to the query-support and attribute filtering would help strengthen the scalability for several visualization aspects (such as the decision impact perspective). Interactivity (v5) and dynamic layout support framework (V13) is also provided by Prefuse. For example, the animated, force-directed layout will continuously change the layouts based on user input and feedback. This enabled me to implement interactive decision visualizations that depend on the attributes of decisions. (For example, decided and approved decisions behave like heavy objects when moved around in the visualization). Fulfilling the other visualization requirements requires additional implementation for the ADDEX tool. With XML decision importing and exporting (V3), decisions can also be represented with other visualization tools, or can retrieve decisions from other sources. The ADDEX visualization can also be customized (V4), such as the user-selectable decision impact

filtering (V11) and decision chronology Y-axis criteria. The ADDEX tool leverages the interactive visualization concepts and capabilities provided by the Prefuse visualization framework for visualization tool usability (V6). However, I acknowledge that tool usability in general could be improved in the next few iterations of the tool. The ADDEX tool briefly addresses adoptability (V7) by allowing the customization of the decision capture approaches and providing a selection of visualization aspects to best address the organization's particular situations and their uses for design decisions. Adoptability and usability are two requirements that are not significantly addressed because of research scope limitations. Tool adoptability and usability are best developed with iterative feedback from the users, so further work is needed to assess the usability and adoptability of the ADDEX tool.

Table 22: Visualization tool requirements matrix

		Decision Visualization Implementation in the ADDEX Tool			
		Tabular listing	Graph'l structr.	Chronology	Impact
Visualization Requirements (Kienle & Müller, 2007)	Quality Attributes	V1: Rendering scalability	Supported	Supported	Supported
		V2: Informat'n scalability	Query dependent	Supported	Supported
		V3: Interoperability	Supported	Supported	Supported
		V4: Customizability	---	---	Supported
		V5: Interactivity	---	Supported	Supported
		V6: Usability	Not addressed in this research scope		
		V7: Adoptability	Not addressed in this research scope		
	Functional requirements	V8: Views	Supported		
		V9: Abstraction	---	Supported	Supported
		V10: Search (Query)	Framework	Framework	Partial
		V11: Filters	Framework	Framework	Partial
		V12: Code proximity	Supported (Decision representation model specific)		
		V13: Automatic layouts	---	Supported	Supported
		V14: Undo/history	Supported (Decision representation model specific)		

For functional visualization requirements, four views (the four visualization aspects) were implemented that look at decisions from an architect's, reviewer's, and maintainer's perspective (V8). In all four visualization tools, decisions were abstracted as nodes in a graph (V9) and were automatically positioned using layout algorithms (V13) to ease the burden of sifting through large amounts of data and promote information scalability (V2). Moreover, the abstraction

conceals non-essential information until the user selects or zooms in on a set of decisions. Decision searching however, was bundled with decision querying (V10) in the previous requirements and a decision querying framework was implemented in the internal structure of the ADDEX tool. Unfortunately, limitations on the time and resources available for the ADDEX development cycle results in deferring the complete implementation of decision searching and querying in the tabular listing and decision structure visualization to the next development iteration. However, for the decision chronology and decision impact visualization, I am able to implement a subset of searching/querying using filtering (V10, V11) to hide decisions or relationships that the users deem to be currently irrelevant so that the users can focus on the decisions that matter to the task at hand. Code proximity (V12) and decision history (V14) are implemented as part of the design decision representation model. Combined with the lightweight decision capture approaches, the visualization provides an entry-point into software code and other artifacts using the captured decision's source links.

CHAPTER 6

EXPERIENCE WITH THE TOOLS

A good way to evaluate a software system implementation is to simply use it. By using the ADDEX tool to capture architectural design decisions during software development and to represent them visually for exploration, we can get a good grasp on how well the ADDEX tool handles actual design decision datasets and supports their exploration through the decision use cases. We can evaluate the ADDEX software system through my personal experience with the tool during its design and development, acquisition of realistic or industry decision sets for the ADDEX tool to represent and manipulate, acquiring feedback on the tool itself by the decision capturers, and observing how someone could use the tool to perform the architectural design decision use cases.

The most significant challenge is that the decision datasets are guarded intellectual properties of their capturers so the confidentiality of the decision sets imposes constraints on who can view and use the captured decisions. This means that using these decision sets for the ADDEX tool evaluation can only be achieved by or alongside the people who or organizations that provided those sets. This would limit the ability to study how people outside a project can learn and manipulate the architecture of the system. However, for the purpose of the experience study, the issue of who performs these use cases is less important than determining the coherence and capability of the ADDEX system to assist people in performing the decision use cases.

6.1 Developmental Self-Testing

I was able to write down my design decisions pertaining to the ADDEX tool during the early stages of its development. Although I have captured many decisions on paper in a notebook, I found that it is often easier to capture design decisions near or within the software development artifact, like the class diagram or software code. Other times, I found helpful software design patterns and architectural guidelines from books and Internet examples, and I often bookmarked

these sources of information during the tool's design phases. These preferences and self-feedback inherently affected the development of the ADDEX tool, which led to my proposal of three capturing approaches for software architectural design decisions and the implementation of customizable capture methods that implement those approaches. The ADDEX tool reflects these capturing methods in the capturing components. Unfortunately, the self-testing is significantly limited by two reasons:

- I am knowledgeable in both the decision capture processes and decision representation model that I defined
- The means to capture my design decisions effectively came at the end of the design

The first reason is straightforward: the author sees only his thoughts and is blind to his own faults. Because I defined the set of requirements and designed the implementations, I became an expert in the system I would like to evaluate, so objectivity is compromised. I address this limitation by consulting with peers in academia and in industry throughout the tool's software development process. I was able to present my ideas and demonstrate my tool in front of researchers and industry practitioners, acquiring useful and practical feedback along the way. The result is a tool that reflects many needs and wishes of both academia and industry.

The second reason is linked to how my idea is developed. To address some of the concerns raised by those in industry and academia, the ADDEX tool underwent many changes. It was difficult to capture many of these decisions using pen and paper (or even a word processor) because of the amount of time required to write them by hand and to keep track of the various decisions. Documenting many of these decisions was deferred until I had more time. As a fast and convenient way to capture these decisions has not been implemented yet, many early architectural decisions were forgotten over time and are lost. However, I was able to capture a limited set of my decisions in my laboratory notebook, e-mails, and software source code. Upon a stable version of the ADDEX tool, I used its three capturing components to gather these decisions and the resulting small set of decisions functioned as a conceptual dataset for the ADDEX tool instead.

The limitations of the self-checking resulted in the need to have people external to my research area capture their design decisions for an actual project they developed. To test the design decision use cases the ADDEX tool is designed to support, we need a project that is sufficiently large and complex to warrant decision exploration tasks like stakeholder risk analysis and decision impact analysis. Large software projects that demand such use cases are usually based in industry. It is clear that we need to acquire industry decision sets from real life development systems to evaluate the ADDEX tool and that the bulk of the decision set acquisition should be performed by another person.

6.2 Decision Acquisition

To demonstrate the ability of the capturing approaches in real-life situations and to gather industry feedback, I presented the ADDEX tool to three industry participants representing separate software organizations and we asked them to capture their architectural design decisions for a project using the tool. The industry participants represent typical developers in software development organizations, and their decisions are actual decision datasets from real-life projects. The objective of this study is to confirm the feasibility and practicality of using the three capturing approaches. I was able to gain supportive feedback regarding the tools and the capturing approaches the methods represent. As well, the participants were kind enough to provide their decision sets for their projects so that I could test the tool's ability to practically capture actual design decisions made in industry.

6.2.1 Industry Participants and Feedback

During the initial contact, all three participants expressed the desire to capture their architectural knowledge and agreed that current capturing processes are insufficient for architectural knowledge. All three organizations used requirements documents and UML for their architectural documentation. A summary of the industry participants are shown in Table 23.

Table 23: Industry participants summary

	Industry	Size	Development	Notes
1	Game Development	Small	In progress, second iteration	Plan for future offshore dev.
2	Information Management	Small	Early design stage	Familiar with knowledge capture
3	Technology corporation	Large	In progress, mature stages	Heavy dev. processes, documentation

The first participant was hired to manage a project already underway in a small 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 was initially involved as a pilot study participant, where the participant provided feedback on how the study is structured and conducted. The participant also contributed feedback on the ADDEX tool. This participant 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 expressed a need to capture architectural knowledge of the system being developed for future reuse, and the participant was actively capturing knowledge and background information on the project. The participant showed enthusiasm for the formal and the top-down tools, but the bottom-up tool was not discussed in detail as the participant's project had not yet entered the detailed design phase at that time. This is a significant reason why they did not want to use the bottom-up approach. The participant did state that the bottom-up approach is interesting and serves its 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 me through meetings and design documents. The participant explained that they could not apply the lightweight capture methods to their situation because most of the architectural design decisions have already been made and code implementation was well underway. Thus, the consensus to use the formal elicitation approach is appropriate for them in this post-design decision capture.

6.2.2 Decision Datasets and Findings

The participants agreed to collaborate with my research by providing me with their project decisions. Due to time and resource constraints of the participants, I elicited decisions from the participants, which is acceptable for a feasibility study on the approaches. In general, I elicited the decisions by listening to the participants as they describe the general architecture and design goals. Then the participants and I discuss what the architectural design decisions are and we document the decisions using one of the decision capturing tools. The decision elicitation also involved revising decisions and creating new decisions and relationships.

The context of my study with the participants does not support an exploration of decision publicity levels (selective-release) due to: 1) the nature of elicitation—personal decisions would not be made known and shared by definition, and 2) the non-disclosure agreements which are in place. Any attempt to investigate publicity levels would result in a single level of publicity – “organization-wide”. Since my goal of acquiring decision sets is to determine how the ADDEX tool handles actual industry data, having the same publicity level for all decisions suffices and would reduce the amount of variables in the study.

The first industry participant was involved as a pilot study participant, so the decisions captured were experimental. After learning about the lightweight top-down capture tool and the types of information to capture, the participant sifted through his own notes he took when he was learning the software project and those decisions were documented using the lightweight top-down capture approach. The participant expressed that the decision candidates from the lightweight top-down capture tool satisfied what was needed without having to create formal decision structures from the decision candidates. This result suggests that tailoring each decision capture

approach to the needs of the organization is important, so it is acceptable if the organization wishes to not complete all of the steps.

The most significant decision set I obtained is from the large technology corporation for a mature project. It was significant because of the size and complexity of the project; the project interfaces with multiple external systems and processes and involves a team of at least 50 software developers taking at least 18 months for development. The introductory volume of the requirements document alone is over 150 pages. To narrow down the scope of the study, I focussed the decision capturing on the deployment configurations and the data model used for this system. In light of the scope reduction, I was able to acquire around 40 decisions through two short, hour-long meetings dedicated to the decision capture and the concept and overview-requirements documents. In each meeting, I listened and discussed the general project architecture as well as the detailed designs of several technical areas. At the same time, the participant and I created as well as revised design decisions and relationships using the formal elicitation tool. By the third meeting, I acquired a total of 52 decisions that pertain specifically to the deployment configuration and data models.

Using the formal elicitation tool, I found that capturing decision rationale from documentation is difficult and I heavily leveraged 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 was difficult if I did not repeatedly ask whether this decision was related to another decision. I found that cross-referencing keywords and scope helped reveal relationships. (I later applied the discovery of cross-referencing keywords to improve the classification and creation of decision impact relationships, which are more general forms of decision relationships.

The decision set that came from the third participant (who was keen on capturing background information and knowledge) provided a good opportunity to use the top-down capturing tool to acquire decisions. I received extensive background documentation, such as statement of work, requirements, email, and other internal assessment documents. From the documentation alone, I

captured 83 decision references in the first iteration of decision capture, of which 62 were 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 were less difficult than with the first decision set, likely due to the availability of background information.

I did not get the opportunity to evaluate the bottom-up capture tool in industry because two of the participants had already started project implementation and did not want to incur more risk by introducing a new step to their development processes when the project is in progress. The third participant had just started preliminary design work on their project and had not developed a base of design artifacts yet for bottom-up decision capture. However, I was able to use the bottom-up capture tool on my own research tool, focussing on the formal elicitation tool source code that contains some decision tags. The tool identified 14 decision tags in the source code and these tags were displayed in the bottom-up capture tool.

6.3 Visualization Study with Industry

A visualization study with industry practitioners and with actual decision sets is needed to evaluate the practicality and reality of using the four visualization aspects for decision exploration. A set of criteria to evaluate decision exploration is to study how people would use the visualizations to perform the design decision use cases. A long-term study with the tool in an industrial setting would be best as it allows for requirements drift, architectural and design decision evolution, and the natural progression of performing the design decision use cases as part of the software development process. However, I encountered some difficulty in finding long-term study participants due to issues of poor timing with participants' projects – the projects are already underway and introducing a new process or tool when development has started carries a certain level of risk. As I could not find any participants willing to perform a long-term study with the ADDEX tool, the visualization study was modified to employ a meeting and casual discussion format. Through scheduled formal meetings, I demonstrated and used the tool with industry participants, and I obtained feedback on the practicality of the decisions. The industry participation is three-fold. I first observed how the participants reacted

when they were shown the tool and I documented their reactions. Then, I acquired actual design decisions from the participants' projects using the tool so that we can visualize them on the tool and explore their design decisions. Lastly, I was able to test how someone could independently use the ADDEX tool to perform the architectural design decision use cases.

6.3.1 Industry Participation

The intellectual property aspects of design decisions significantly affect the study. The established confidentiality agreements with the study participants limits who I can share the information with for further study, which results in the need for the original design decision set donors to continue with the study. Reusing study participants has both benefits and drawbacks; reusing participants reduces the learning curve and the amount of redundancy in acquiring another decision set for the visualization study. However, the reduced learning curve biases the results toward the "expert" decision capturer. In this particular experience study, we will look at the decision exploration capabilities of the ADDEX tool, so the effects of learning are negligible. Unfortunately, I have not found an organization or person working on open-source projects willing to participate in the research in time for the study.

After the decision capture study with the industry participants, I asked them whether they would like to continue the collaboration to help me investigate the visualization concepts. Due to the poor timing and limited time availability, two of the original study participants declined the study; however, the third participant agreed to participate. The participant is a senior software engineer at a large technology corporation. (Refer to the third industry participant in Section 6.2.1, summarized in Table 23 on page 88.) This corporation is both process and documentation heavy. The participant was involved in a multi-national project to develop an elaborate modeling system. This system interfaces with various global databases frequently to stay updated, but the system is constrained by many domain-specific standards and protocols. The decision set the participant captured for the study focussed on the deployment configurations and the data model of the system being developed. (Shortly after the study commenced, the participant invited a senior software developer working on the project to join in on the discussion. For simplicity, I will refer to both participants as a single participant.)

6.3.2 Feedback

After demonstrating the tool to the participant using their own decision dataset, I asked the participants what their impressions of the ADDEX visualization component were. They found that the decision and relationship lists were acceptable, but could not comment much about the lists besides an implementation detail of whether the listed items could be filtered or sorted. For the graphical decision structure visualization, the participant stated that the decision identifier used in the default zoom-level is not very intuitive, as it can be hard to mentally map decision details to the decision identifiers. Although the semantic zooming offers additional decision information if the user zooms in on a decision, the participant found that it is somewhat difficult to reference decisions without referring to other views. The participant found the decision relationship graphs to be interesting, but the participant also reported that the explicit decision relationships are difficult to elicit and categorize, partly due to the various relationship definitions and the tacit nature of defining these relationships. The decision structure view enables the participant to realize an earlier documented decision had been deferred to a later release, and the decision set is updated accordingly.

For the decision chronology view, the participant commented that it is useful to see decision-making sessions, and they found the “author” criterion for the user-selectable y-axis to be an interesting application. However, for the decision impact view, the participant felt that the decision impact view suffers some functional usability because the coarse-grained filtering resulted in a diagram that has too much interconnectivity, and suggested that implementing a user-defined query mechanism would help with the readability and usability of this view. This reaction is expected, as the implementation of fine-grained filtering was not complete at the time of the study. Yet the participant expressed that the decision impact view could be effective in identifying decisions that could be indirectly impacted with further filtering improvements. The participant also said that he can see himself using this visualization aspect as a part of his “design analysis toolbox”.

6.4 Tool Usability

Usability is an important factor to consider when evaluating the integrated solution through its tool implementation. Specifically, how well the integrated solution assists people in capturing

and exploring software architectural design decisions depend on how well people can use the ADDEX tool to capture and explore software architectural design decisions. This can be evaluated by studying how people would use the tool to perform decision use cases. The testing should be in the form of a usability study, where the results of the study would be used in the next development iterations of the ADDEX tool. As I have acquired feedback already on the decision capture and visualization concepts implemented in ADDEX, the goal of the first usability study is to confirm whether the difficulties identified earlier by the industry participants above are consistent or common. This would give a solid baseline and direction for improving the ADDEX tool. Detailed, statistically accurate participant sampling would not be appropriate at this time. Themes and patterns can be determined and the ADDEX tool can be further refined to handle confusing or incorrectly performed tasks. The first ADDEX usability study is a pilot study, in which I perform the study on one or two participants so that the study results feed directly back to the next development iteration of the ADDEX software tool.

As software architectural decision capture and exploration are valuable in cases where the software project is large or complex, capturing and exploring small, fictitious decision sets with the ADDEX tool seem trivial or contrived. For a realistic usability study, a set of software architectural design decisions provided by industry of actual software projects is preferred. However, the limited timeframe of my thesis research and the limited dataset audience imposed by the dataset confidentiality agreements made it difficult to find a usability study participant to assess the usability ADDEX with industry data if the original decision set donors are unavailable. As an alternative, I planned to use a decision set that came from the ADDEX tool in lieu of industry decisions sets. However, because I designed the ADDEX tool and I captured the decisions myself, the resulting decision set may reflect what I subconsciously want people to see or not see, so using the ADDEX tool decision set would likely be biased (refer to Section 6.1). On the other hand, since I am studying how people would interact with the tool to perform decision use cases, the information bias of the decision set is of lesser importance than on the tool implementation.

A chance circumstance allowed me to find a software developer in the same project as the previous participant to independently use the ADDEX tool. This occurred several months after

the initial dataset acquisition with the previous participant. The new participant is a recent university graduate from a software engineering program and the participant has worked on the project close to a year. Although the participant implemented software code for the system, the participant has not worked on many areas of the system and does not know much about the decisions behind the project's software architecture.

In this study, the participant has access to a computer with the ADDEX tool open and running with the organization's design decision set loaded. On the table are a pen and several sheets of blank paper. I gave the participant a verbal explanation of what an architectural design decision is, a brief walkthrough of the ADDEX user's guide. The participant was given a few minutes to play around with the tool and ask any questions about the tool or the concepts for the study before beginning. The participant is asked to perform three tasks on the project's decision set, which contains over sixty decisions. I sat approximately one meter behind the participant (and in such a way where I can also see the computer screen) and documented the participant's actions. The participant understands that there are no time constraints so he could take as much time as he wants to perform the task. I kept a time log of the events in order to record a reference point of activity for comparison across studies at a later time. At the end of the study, I discussed the tasks with him. This study session took approximately forty minutes to perform. The results of this study session are summarized in Table 24.

6.4.1 Performing the Tasks

The three tasks are based on the decision use cases listed in Table 16. To reduce the amount of time a participant needs to commit for the study, only the use cases I classified as high and medium priority are used. The first task I asked the participant to perform is to find out if there are any architectural design decisions that constrain other decisions (use case U2), and if there is at least one, choose one and describe why that decision constrains the other decision (use case U4). The participant started the tool and went to the tabular list view and scrolled through the list of all decisions for several seconds, before the participant selected the decision structure view. The participant then panned through the graph of decisions and found eight decision relationships where decisions constrained other decisions. The participant wrote down the related decision identifiers and double-clicked on one of the constraining decisions to open the decision

and wrote down the epitome of the decision. Two minutes had passed when the participant identified all eight decisions. The participant viewed several decisions of interest and clicked on the relationships to view the relationship comments. Another two minutes passed before the participant realized the task is complete, at that point the participant explained the rationale of one of the constraining decisions and the relationship comment to me, stating that those were the reasons for constraining the other decision. This task was completed in about four minutes.

Table 24: Sequence of actions performed for certain tasks

Task	Duration (min:sec)	Sequence of actions	Associated use cases
Find constraining decisions & describe why one decision constrained the other	3:55	<ul style="list-style-type: none"> ▪ Opened decision list view ▪ Briefly scrolled/viewed list of design decisions ▪ Opened decision structure view ▪ Searched for “constrains” relationship, found 8 ▪ Wrote down decision identifiers (ID) of relationship ▪ Picked one constraining decision/relationship ▪ From structure view, viewed constraining decision ▪ Also from structure view, viewed relationship ▪ Identified reason from the relationship’s comment 	U2, U4
Find changed decisions (last 2 months) & determine dependencies	6:15	<ul style="list-style-type: none"> ▪ Opened up decision impact view ▪ Switched to decision chronology view ▪ Viewed timeline for entire period (6 months) ▪ Noted 5 decisions in the last 2 months ▪ Copied down the decision IDs of those decisions ▪ Viewed details of those five decisions in this view ▪ Opened decision list view ▪ Selected an identified decision ▪ Clicked “relationships” to see relationships list ▪ Found and viewed relationships for those decisions ▪ Declared there are 2 dependencies 	U1, U3, U5
Assess the design for design-implementation disparity & add decision explaining differences	9:40	<ul style="list-style-type: none"> ▪ Switched among the four visualization views ▪ Expressed chronology view “won’t be of much use” ▪ Went to decision list view ▪ Scanned through all decisions in list, found two decisions in the area of concern ▪ Switched to impact view ▪ Enabled the decision impact relationship filter for “scope”, disabled all other impact relationship filters ▪ Clicked one of the identified decision—the decision moved to the centre of the screen. ▪ Hovered mouse over the decision, found three immediately impacting decisions (by scope) ▪ Wrote down decision IDs of the 3 other decisions ▪ Switched to decision list view ▪ Read through the decisions’ details closely ▪ Identified decision conflict. ▪ Logged in via the decision list view interface ▪ Created new decision to resolve decision conflict 	U6, U7

The next task I asked the participant to perform is to look up which decisions were changed in the last two months (use cases U1 and U5) and determine whether there are other decisions that depend on those created decisions (use case U3). The participant first opened the decision impact view, but switched to the decision chronology view several seconds later to view the timeline of design decisions. This timeline spans the entire period of the decision capture process (over six months). There were five changed decisions in the last two months. The participant moved the mouse cursor over each of the five decisions and wrote down the five decision IDs. The participant viewed the details of these decisions in the chronology view, and then opened the decision list view, selected one of the identified decisions, and immediately brought up the relationships list. However, the participant voiced that he expected to find only the relationships that involve the selected decision. At this point, two minutes forty seconds have passed since the start of the task. The participant then scanned through the relationship list for the five affected decisions on either side of the relationships. For each of the three relationships found involving the concerned decisions, the participant viewed and studied the relationship and he declared task completion shortly after viewing those relationships. Total time for this task was six minutes fifteen seconds.

The third task required the participant to be critical with the design decisions to look for inconsistent, overly general, or confusing decisions or relationships and tidy up the system (use case U7). If applicable, the participant should determine whether the decision set agrees with what the participant already knows about the project (as the decision set in this study came from the participant's own project). The final action the participant needs to perform is to add a decision (use case U6) to clarify or correct the design. The participant started this task by looking at the decision structure view that he left open from the previous task, and began to bring up the other three visualizations. He focused his attention on the decision list view, browsing the decision epitomes and viewing the details as he works his way down the list of decisions. Halfway down the list and approximately four minutes into the task, the participant asked for further information about the task by requesting whether there is a specific area of interest. I replied with the suggestion that "there were reports of performance issues related to large memory usage involving the <scope area>". The participant immediately found one of the

decisions in the scope area, wrote down the decision ID, and brought up the decision impact view. The participant filtered the decision impact relationships to display only the “scope” impact relationships, and then he found the identified decisions and clicked on it. The decision moved to the centre of the screen while the layout rearranged in animated sequence around the centre decision. The participant rested his mouse cursor over the centre decision to highlight the decisions that share the same scope. In less than a minute, the participant identified three additional decisions and he went to the decision list view to study the three newly identified decisions. By the six-and-half minute mark, the participant voiced his thoughts that there is a conflict between two of those decisions; one decision describes a system behavior that contradicts the desired outcome of the other decision. Using the decision list view, the participant logged into the system and created a new decision to resolve the conflict between the two decisions by adding a decision to use a different system. The participant filled out all the fields in the decision dialog, selected a category and left the change log comment empty when saving the decision. He then declared task was complete. It took a total of nine minutes forty seconds for this task.

6.4.2 Observations and Analysis

The first task is where the participant needs to find and explain a decision that constrains another decision. Although the participant started out with the decision list view, the participant used the decision structure view to find the decision relationships, lending support to the design intent of the decision structure view. The participant quickly figured out that decisions can be viewed by clicking on the decision of interest and then clicking the “view” button. However, the participant took a little more time to view relationship details, as it was not immediately apparent to click on the relationship in order to view the relationship details. As this was the participant’s first task, the participant took some time to read through several decisions in greater detail that were not directly related to the task. When I asked the participant about it afterwards, the participant stated that he was curious about some of the other decisions and took some time to view them.

The second task involved finding which decisions were changed in the two months and determining the dependencies for those decisions. The participant initially went to the decision impact view to find dependencies, but realized that he did not know which decisions were

changed easily in that view. The participant then asked for a task clarification on the definition of a “changed decision”, that is, whether correcting a spelling mistake or changing the state of a decision is a decision change. My response was a “change” includes any modification to the decision information, including spelling or state change. Using the decision chronology view, the participant was able to find which decisions have changed in the past two months and he identified the decision IDs by resting the mouse cursor over the decisions corresponding to the timeline region and writing down the decision IDs displayed on the mouse-over text. The participant did not zoom in to narrow down the viewed time frame, perhaps because there were only a few changed decisions and they were spread over the two months. The second half of the task (finding dependencies) took a different approach and took longer than I expected. The participant first brought up the decision list view and clicked on one of the identified decisions with the intent on bringing up all the relationships associated to that design decision, but the ADDEX implementation listed all the decision relationships for all decisions instead. The participant chose to continue with this large list of relationships and scanned through the list to find five relationships that involve those decisions. Then the participant viewed each of those relationships and thought about them for several minutes to work out the logic behind the decisions before he decided that there were two dependent changes for one of the changed decisions. This task is surprising in several ways. The participant did not use the decision structure view nor continued with the decision impact view (that he started to do at the beginning of this task), which would have made identifying decision relationships clearer. The participant also noted that he expected the relationships list to be automatically filtered to show only relationships pertaining to the decision of concern, which is logical and is unfortunately an uncaught usability oversight in the ADDEX implementation. As well, identification of a changed decision is not very clear to the participant; the participant noted that the change logs should have an automatically-generated “what’s changed” column that identifies which values were changed from the previous version.

The last task involved critically assessing the design as represented by the decisions, looking for confusing or inconsistent decisions between other decisions as well as between the design and implementation. The participant was required to create a new decision that would explain any disparity between the decisions and/or implementation to tidy up the system. This task gave the

participant significant freedom to interpret and use the ADDEX tool independently. To understand why the participant performed the actions I documented, I reminded the participant to describe his thought processes when he was performing the task. At the beginning of this task, I observed the participant going back and forth between the four different visualizations. I inquired about this afterwards, and the participant replied that he didn't know where to begin. The participant also said that he went from visualization to visualization to "get a feel" for the decisions, and then he decided to start with the decision list view to gauge how correct the decision contents are. The participant commented that many design decisions he browsed through were new to him, but he sees how the decisions are mapped to the product he is currently implementing. A point of interest is that the participant did not immediately find any inconsistencies or design discrepancies without requesting a specific area to focus on. This is understandable and expected, as the decision set included over sixty decisions that cover a range of decisions, and it is unlikely that someone who is not familiar with these decisions would become instantly aware of all of them. However, once I gave the participant a specific area to look for, the participant immediately found a decision related to it and was able to find the other decisions in the area quickly and eventually identified a conflict between two decisions. It is also interesting that the participant used the decision impact view to find other decisions of the similar scope. This was unexpected, as I originally thought the decision list view would be more suited for this activity, but it seems that using the impact view is just as effective.

The final part of the task involved adding a decision. The participant created the decision from the decision list view. The capturing approach is formal elicitation, as the decision had to be documented directly and any decision details to be entered all at one time. The participant filled out all the fields in the decision creation dialog, but the participant opted not to enter a change log comment before saving his decision. Moreover, the participant did not create any decision relationships that link the two concerning decisions together with this newly created decision before declaring his task to be complete. When I asked the participant why he chose not to add relationships from his decision to the other two design decisions, he replied that he was focussed on finishing his task that he forgot to step back to look at the big picture. This comment reminds us of the difficulties in capturing design decisions, as often times people become involved with the task at hand that they often forget about the decision capture completely.

Although the participant performed three tasks, there is actually another task I planned, but it was not included in this participant study. The task involved spotting the subversive and critical stakeholders (use cases U8 and U9). It was not included because of the way the decision set has been captured. Although the ADDEX tool explicitly supports multiple authors to represent various stakeholders, spotting the subversive and critical stakeholders is difficult to perform in this study because the decision dataset was created from the elicited-view of one person. Decision authorship in the tool is based on who was entering the decisions, so it is possible to have multiple authors but one stakeholder/viewpoint, as in the case of this decision dataset. Therefore, the remaining two prioritized use cases were excluded from the participant study as it requires prior support for the stakeholders' interests during the decision capture process.

6.4.3 Tool Refinements

The tool usability study provides significant insight on how people use the ADDEX tool and what areas require further refinement. Through the study, I identified several areas where the tool can be refined. First, I noticed that the participant frequently flipped back and forth between different views. Furthermore, the participant often wrote down the decision identifiers found in one view on a piece of paper and then searched for them immediately in the next view. One way to remedy this situation is to implement decision cross-referencing between visualization views. For example, selecting one decision in one view would select the same decision in another view, and this would eliminate the need to search for design decisions between views. The limited decision query support can also be improved to facilitate easier decision searching and navigation. One query support improvement would be supporting key-word-based searching (to parse through fields like the “epitome” or “rationale”), as well as providing comparative queries (such as “all decisions with two or more relationships”). In the decision chronology view, adding previous decision versions to the visualization and linking them together with versioning traces can address the difficulty in identifying decision changes and determine the state of the design at a specific period in time. There is already a supporting framework built into the ADDEX tool for these usability improvements. Another improvement would be adding a “legend” for the decision chronology view to help users map decision states to the node shapes in the view.

Interestingly, unlike the participant in the visualization study earlier, this participant found the semantic-zooming feature to be very useful for him.

Some other usability issues were made apparent after the ADDEX tool usability study. The most significant issue is that the default settings for the decision impact view would typically overwhelm a user with information. If the defaults of the decision impact view were set to display only one type of impact relationship and renders the impact relationships to a single degree, then people would be less intimidated or overwhelmed with information. Combined with the cross-referencing improvement discussed above, the decision impact for a decision of interest can be automatically selected. The usability study also helped identify small changes to improve user convenience, such as showing only decision relationships pertaining to a selected decision, including decision IDs in all decision views, and implementing automatic change log generation to document which values have changed.

The study provided the necessary motivation to refine the tool. The first iteration of the ADDEX tool focussed on the tool's functional implementation and the technical challenges to implement such a tool. This pilot study on the tool's usability brought insight to how the tool implementation can both support and hinder decision capture and exploration. The study also confirmed the areas where the ADDEX tool needed to be changed. I support that usability studies should be performed alongside functionality implementation so that the findings and results from the first usability study would feed directly into the next development iteration of the software tool.

6.5 Decision Capture Tool Comparison Experiment

Recently, a fellow graduate student performed an independent decision capture tool experiment with several industry practitioners (Ting, 2009). Ting commenced this experimental study around the same time as my own industry evaluation of the ADDEX tool. The experiment focussed on comparing three decision capture tools in industry. These three tools are the ADDEX decision capture tool, the ADDSS tool, and Compendium. For the ADDEX tool, the formal elicitation component was used for the study.

6.5.1 Experiment Overview

Ting's experiment attempts to determine which decision capturing tool is more mature for use in industry and identifies the strengths and weaknesses in the three tools. The experiment consists of performing three decision capturing sessions of around fifteen minutes in duration for each tool. The first capture session involves using the tool to capture decisions without instructions as to how to use the tool or what a design decision entails. The second session starts after a brief information session explaining the features of the tool and introducing the concept of decision structures. The last session narrows down decision capturing to a specific component in the project for a more controlled comparison of decision capture across three tools.

The experiment involves three industry participants with three, five, and eleven years of software development experience. All of the participants are software engineers who previously worked on a software project together. They shared the same roles as both designer and developer for the project, and none of them reported themselves as aware of the concept of design decisions before. The participant with three years of software development experience was assigned to the Compendium tool, while the participant with five years experience was assigned to the ADDSS tool. The participant with eleven years of experience was assigned to the ADDEX tool.

6.5.2 Experiment Results

Ting's experiment showed that the Compendium tool is most ready for industry use in terms of usability and functionality, followed by the ADDSS tool and then the ADDEX tool. This result is not surprising, as the ADDSS tool debuted three years ago and is already in its second version, while the Compendium tool was introduced in 1996 and has been developed, maintained, and made publicly available for more than a decade. When comparing overall decision capture rates, the Compendium tool captured decisions at a faster rate than the other two tools in the first two sessions (over 50% faster in the second session) and captured the most decisions overall. Interestingly, the ADDEX tool had the highest rate of capture during the focussed decision capture session. Overall, the ADDSS tool and the ADDEX tool are similar in that they captured approximately the same amount of decisions, but the ADDEX tool captured more decisions than the ADDSS tool during the second and third sessions. From the results, the ADDEX tool has a high learning curve, resulting in the lowest decision capture rate during the first session but the

highest in the third session. Further experiments after improving ADDEX tool usability could determine whether the higher capture rate in the focussed capture session can be attributed to the tool's functionality or increased user familiarity.

The experiment also highlighted the strengths and weaknesses of each of the three tools from the participants' perspective. Ting reports that the Compendium tool is easy to learn with a low learning curve, and handles high-level design dependencies well, but lacked some support for representing decisions as first-class entities and modelling of decision states, like "rejected" decisions. The ADDSS tool is also easy to learn and captures high-level concepts well, and supports design planning by enforcing the concept of iterations, but details and complicated ideas are difficult to express due to input and selection limitations, which may hinder its effectiveness during the later phases of a project. The lack of diagram support makes textual explanations of certain concepts difficult. Despite that only the formal elicitation capturing component is used for the study, the ADDEX tool is identified to be a good reference tool and the decision exploration components are useful. Ting concludes that it has potential, but the tool suffered the most from its relatively immature state. The ADDEX tool's lack of an online "help" functionality and its need for some user-interface usability refinements and features ultimately hinders users from capturing decisions effectively. Some missing features include external file linking in the formal elicitation capture component and a cleaner decision structure layout algorithm that will not clutter the visualization with excessive decision or relationship crossings.

In terms of decision quality, some of the captured decisions from the Compendium tool were obvious and could be found in the requirements document, but this could be attributed to the participant having the least software design and development experience among the participants in the study. The differences in the tools' decision quantity and quality could also be attributed to the fact that the three participants may not have participated equally in the design of the selected component to document for the third capture session. Having the three participants each evaluate the three tools would reduce the effects of design expertise when comparing decision capture rates and decision differences. All three tools promoted the importance of decision capture during software development and facilitate decision capture and exploration in industry.

CHAPTER 7

CONCLUSIONS AND SUMMARY

The work of this thesis focuses on summarizing and integrating the current works involving software architectural design decisions. A frequently mentioned challenge is that it is still difficult to capture architectural design decisions and convey them to other software developers; moreover, it is also difficult to explore the captured decisions effectively. A holistic, system-based tool is clearly needed to address the interdependencies of decision exploration and capture. The creation of the ADDEX tool attempts to address this cyclical relationship and attempts to integrate all the common goals, guidelines, requirements, use cases, and challenges currently identified by many researchers and industry practitioners. The ADDEX tool addresses the decision capture problem by supporting three customizable decision capture processes that can be tailored to the specific capturing needs of the organization. In addition, the tool addresses the visualization requirement by implementing four visualization aspects that support the identified use cases. ADDEX attempts to represent the common vision of what a design decision support environment should be for software organizations to capture and explore design decisions. The implemented ADDEX tool was brought before industry experts for feedback and industry datasets were used to test the practicality of the tool-based solution that ADDEX implements. This chapter reviews the contributions of the work within the software engineering community.

7.1 Research Goals Summary

The goal of this thesis is to integrate the recommendations of several research contributions to determine a tool-based solution for software organizations to capture and explore their architectural design decisions. This solution reflects the current views common to the researchers and industry practitioners in the field of software architecture and maintenance. This goal is achieved by looking at the current works in literature to determine the common decision representation model and identifying common challenges, issues, and implementation requirements for a tool-based solution. The practicality of the integrated solution is evaluated

through the implementation of a design decision system tool and through the use of actual industry datasets acquired and represented with the tool. Useful industry feedback about the implemented solution is gathered at the same time.

7.2 Contributions of This Work

I have integrated the works of many researchers and industry practitioners to summarize the collective state of software architectural design decision support systems, and I made recommendations on how we could meet some requirements and objectives through a tool-based solution. I have shown that we can implement such a tool (i.e., ADDEX) and I have identified that the tool can support actual industry datasets. I demonstrated the tool to industry practitioners and gathered feedback about the tool.

The following is a summary of my contributions:

- A solution that integrates the current common issues, challenges, requirements, use cases, and guidelines to capture and explore design decisions using a system-based tool. This solution represents the current state of research in the software architecture and maintenance communities
- A proposal of using three capture approaches (together or separately) to encourage and facilitate decision capture:
 1. formal elicitation,
 2. lightweight top-down, and
 3. lightweight bottom-up
- A proposal of four visualization aspects that apply to software architectural design decisions to promote decision exploration:
 1. tabular lists,
 2. decision structure visualization,
 3. decision chronology visualization, and
 4. decision impact visualization
- A tool called ADDEX that implements the integrated solution for software architectural design decision capture and exploration
 - Combines decision capture and exploration using a holistic approach

- Supports decision capture across various stages of the development process
 - Supports capture of incomplete decision information
 - Visualizes four aspects of design decisions to support decision exploration
- A demonstration of the tool and the integrated solution it represents to capture and represent actual decision sets acquired from industry
- An evaluation of the tool by four industry practitioners to gain feedback on the tool-based solution and the proposed decision capture approaches and decision visualization aspects

7.3 Future Work

In addition to identifying and implementing a set of requirements for software architectural design decision systems, there are other areas of further research in the tool-based support of design decision capture and exploration. Two significant areas for future work are implementation and evaluation. Implementation plays a large role in the users' experience with the software tool as the experience is governed by how well the tool is implemented. Accurate and effective evaluation is needed to determine the successes and shortcomings of the research work.

Improved tool implementation is necessary to reduce the effects of the implementation on the study results. The result of Ting's experiment emphasizes the importance of the maturity of a tool's design and implementation. Tool usability is a priority; satisfying this requirement would help users capture and explore design decisions without the tool getting in the way of their tasks. Improved functionality like decision querying and filtering in the ADDEX tool's visualization component should enable people to perform what they want to do with the decisions using the tool. As well, implementing better methods to cross-reference keywords and decision attributes could also help identify other potential decision impact. Improving default values and settings for all aspects of the tool will help, and providing external file linking or storage could save users from additional data entry. For decision capture, the three decision capture approaches are linked to how well the implemented capturing tools are integrated into the processes of an organization or the daily routines of a software architect. In lightweight top-down capture, the capture tool should be implemented as a part of the daily tool set, such as a word processor plug-in, a design tool add-on, or an e-mail client extension. The development

and use of closely-integrated decision capturing tools should be investigated to determine the effects of the three capturing approaches on an organization's decision capture process. In both decision capture and decision visualization, the ADDEX tool is my personal interpretation of the requirements, so it would be interesting to see how other researchers and industry practitioners would develop their decision capture and exploration tool using the same or similar set of requirements identified in this thesis. Explicitly implementing the concept of publicity levels by linking the selective release of design decisions to groups of users in all four components of the ADDEX tool would lead to an interesting area of further study. Supporting decisions that are intentionally left tacit or implicit may also yield interesting results. General improvements to tool usability would help lower the effects of the tool on the study. For example, several users have browsed across visualization aspects, so cross-referencing (by highlighting) design decisions across the various open visualization views could help. Also, the "history" functionality of the ADDEX tool can be extended to include the "Undo" concept, as every decision addition and manipulation is logged and tracked. Using other visualization layout algorithms could also improve usability by reducing clutter or cross-placement of nodes and edges in the visualization.

Although I am showing that it is possible to implement a tool using the set of integrated requirements, we still need to perform an evaluation on how well the tool implements those requirements. With the limited industrial evaluation of the lightweight bottom-up decision capture component, the first recommendation is to acquire non-trivial industry datasets using the bottom-up approach to determine its decision capturing effectiveness in real industrial situations. The industry feedback on the ADDEX tool suggests that the work is on the right track, but a detailed evaluation of the tool and the proposed capture approaches and the visualization aspects is necessary to determine how useful the requirements and guidelines are in practice. We should perform this detailed evaluation in the next iteration of the research work. Performing additional usability and long-term field studies (six or more months) with the tool could also allow us to determine and improve the usability and adoptability of the visualization tool. Further usability studies also help customize the implemented visualization aspects to the specific needs of individuals and organizations. Ting's experiment compared the ADDEX tool against other design decision tools and found some strengths and limitations of the tools. However, we should perform another empirical study after making the necessary changes to address the concerns and

issues raised in the first study so that we can better evaluate and understand the proposed requirements and use cases represented by the ADDEX tool. We should also deploy ADDEX in industry after refining my tool's implementation to evaluate how well the tool handles the decision capturing processes and decision exploration within an organization and to determine what other capabilities industry practitioners require for software architectural design decisions. Performing a general study on the effects of various levels of disclosure for design decisions could also help answer interesting questions, such as how much personal decisions influence the end design.

7.4 Conclusion

This thesis describes the issues, challenges, requirements of capturing and using architectural design decisions during the software development process. The thesis integrates the works of various researchers and industry practitioners to arrive at a tool-based solution that tries to satisfy many of the guidelines and recommendations regarding the capture and exploration of software architectural design decisions. As the issues of decision capture and decision exploration are interrelated, the tool-based solution should take a holistic approach. To assess the practicality of the tool-based solution I created the ADDEX tool that combines both decision capture and decision exploration together in a common environment. I proposed three approaches to decision capture (formal elicitation, lightweight top-down and lightweight bottom-up capture) to address the specific needs and situations that various software organizations have for architectural design knowledge, and I proposed four decision visualization aspects to assist these organizations to use the design decisions as described by various researchers. These proposals are reflected in the implemented ADDEX tool.

However, we need to verify that the developed tool meets the common guidelines and requirements of the tool-based solution that I brought together from current research works in the software architecture and maintenance communities. The capture and representation of actual industry decision sets using the developed tool demonstrates that it is possible to implement the set of goals, requirements and design decisions combined from various researchers and industry practitioners. The industry decisions sets and evaluation from industry participants also helped evaluate whether the tool (and to some degree the general tool-based solution it represents) met

the goals and challenges described in the works of those I used to integrate together. Analyzing the industry feedback and decision datasets also allows us to see how the proposed capturing approaches and visualization aspects can be improved to better support decision capture and exploration of software architectural design decisions. We should perform further studies to study and improve the decision capture process in the light of improving the usability and exploration of design decisions for software organizations. Since we have come to a collective consensus of what an architectural design decision system should constitute, we should focus on decision capture processes and decision exploration through visualization and continue the research into how decision capturing processes and decision visualization can be improved to investigate the decision capture and exploration potential in the current and future works involving software architectural design decisions.

REFERENCES

- Abrams, S., Bloom, B., Keyser, P., Kimelman, D., Nelson, E., Neuberger, W., Roth, T., Simmonds, I., Tang, S. and Vlissides, J.: Architectural thinking and modeling with the Architects' Workbench. *IBM Systems Journal*, 45(3) pp. 481-500 (2006)
- Akerman, A. and Tyree, J.: Using ontology to support development of software architectures. *IBM Syst. J.*, 45(4) pp. 813-825 (2006)
- Avgeriou, P., Kruchten, P., Lago, P., Grisham, P. and Perry, D.: Architectural knowledge and rationale: issues, trends, challenges. *SIGSOFT Softw. Eng. Notes*, 32(4) pp. 41-46 (2007)
- Babar, M.A., Gorton, I. and Jeffery, R.: Capturing and Using Software Architecture Knowledge for Architecture-based Software Development. In: Proc. 5th International Conference on Quality Software (QSIC), pp. 169-176, Melbourne (2005)
- Babar, M.A., Gorton, I. and Kitchenham, B.: A framework for supporting architecture knowledge. In: Dutoit, A.H., McCall, R., Mistrik, I. and Paech, B. (eds.): *Rationale Management in Software Engineering*. Springer-Verlag (2006) pp. 237-254
- Bosch, J.: Software Architecture: The Next Step. In: Oquendo, F., Warboys, B.C. and Morrison, R. (eds.): In: Proc. European Workshop on Software Architecture (EWSA 2004), vol. LNCS 3047, pp. 194-199. Springer, Heidelberg, St Andrews, Scotland (2004)
- Bratthall, L., Johansson, E. and Regnell, B.: Is a Design Rationale Vital when Predicting Change Impact? A Controlled Experiment on Software Architecture Evolution. In: Proc. Second International Conference on Product Focused Software Process Improvement, pp. 126-139 (2000)
- Bruegge, B., Dutoit, A.H. and Wolf, T.: Sysiphus: Enabling Informal Collaboration in Global Software Development. In: Proc. First International Conference on Global Software Engineering, pp. 139-148, Costao do Santinho, Florianopolis, Brazil (2006)
- Burge, J.E. and Brown, D.C.: Reasoning with design rationale. *Artificial Intelligence in Design '00*. Kluwer Academic Publishers, Netherlands (2000) pp. 611-629
- Burge, J.E. and Brown, D.C.: Design rationale for software maintenance. In: Proc. 16th IEEE international conference on automated software engineering (ASE'01), pp. 433-436 (2001)
- Burge, J.E. and Brown, D.C.: An Integrated Approach for Software Design Checking Using Rationale. *Design Computing and Cognition '04*. Kluwer Academic Publishers, Netherlands (2004) pp. 557-576
- Burge, J.E. and Brown, D.C.: Rationale-based Support for Software Maintenance. In: Dutoit, A.H., McCall, R., Mistrik, I. and Paech, B. (eds.): *Rationale Management in Software Engineering*. Springer-Verlag Berlin Heidelberg (2006) pp. 273-296
- Capilla, R., Nava, F. and Dueñas, J.C.: Modeling and Documenting the Evolution of Architectural Design Decisions. In: Proc. Second Workshop on SHAring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent, pp. 9-15. IEEE Computer Society, Minneapolis, MN, USA (2007)

- Capilla, R., Nava, F., Pérez, S. and Dueñas, J.C.: A web-based tool for managing architectural design decisions. *SIGSOFT Software Engineering Notes*, 31(5) (2006)
- Conklin, J. and Begeman, M.L.: gIBIS: a hypertext tool for team design deliberation. In: Proc., pp. 247-251. ACM, Chapel Hill, North Carolina (1987)
- Conklin, J. and Burgess-Yakemovic, K.C.: A process-oriented approach to design rationale. *Design Rationale Concepts, Techniques, and Use*. Lawrence Erlbaum Associates, Mahwah, NJ (1996) pp. 393-427
- de Boer, R.C. and Farenhorst, R.: In Search of ‘Architectural Knowledge’. In: Proc. Third Workshop on SHAring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent, pp. 71-78. IEEE Computer Society, Leipzig, Germany (2008)
- Deelen, P., van Ham, F., Huizing, C. and van de Wetering, H.: Visualization of Dynamic Program Aspects. In: Maletic, J.I., Telea, A. and Marcus, A. (eds.): In: Proc. Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007. 4th IEEE International Workshop on, pp. 39-46, Banff, Canada (2007)
- Dueñas, J.C. and Capilla, R.: The Decision View of Software Architecture. In: Proc. 2nd European Workshop on Software Architecture, vol. LNCS 3527, pp. 222-230. Springer Berlin / Heidelberg, Pisa, Italy (2005)
- Dutoit, A.H., McCall, R., Mistrík, I. and Paech, B.: Rationale Management in Software Engineering: Concepts and Techniques. In: Dutoit, A.H., McCall, R., Mistrík, I. and Paech, B. (eds.): *Rationale Management in Software Engineering*. Springer-Verlag Berlin Heidelberg (2006) pp. 1-48
- Dutoit, A.H. and Paech, B.: Rationale-based Use Case Specification. *Requirements Engineering Journal*, 7(1) pp. 3-19 (2002)
- Falessi, D., Cantone, G. and Becker, M.: Documenting design decision rationale to improve individual and team design decision making: an experimental evaluation. In: Proc. 2006 ACM/IEEE international Symposium on international Symposium on Empirical Soft. Eng. ISESE '06, pp. 134-143. ACM (2006)
- Falessi, D., Cantone, G. and Kruchten, P.: Value-Based Design Decision Rationale Documentation: Principles and Empirical Feasibility Study. In: Proc. Software Architecture, 2008. WICSA 2008. Seventh Working IEEE/IFIP Conference on, pp. 189-198. IEEE Computer Society, Vancouver, BC, Canada (2008a)
- Falessi, D., Capilla, R. and Cantone, G.: A value-based approach for documenting design decisions rationale: a replicated experiment. In: Proc. 3rd international workshop on Sharing and reusing architectural knowledge, pp. 63-70. ACM, Leipzig, Germany (2008b)
- Farenhorst, R., Lago, P. and Van Vliet, H.: Effective Tool Support for Architectural Knowledge Sharing. In: Oquendo, F. (ed.): In: Proc. First European Conference on Software Architecture (ECSA 2007), vol. LNCS 4758, pp. 123-138. Springer-Verlag Berlin Heidelberg, Aranjuez, Madrid (2007)
- Fischer, G., McCall, R. and Morch, A.I.: JANUS: integrating hypertext with a knowledge-based design environment. In: Proc. Second annual ACM conference on Hypertext, pp. 105-117. ACM Press, Pittsburgh, Pennsylvania, United States (1989)
- Gotel, O. and Finkelstein, A.: Contribution Structures. Proceedings of 2nd International Symposium on Requirements Engineering RE95, pp. 100 - 107 (1995)

- Grant, K.A.: Tacit Knowledge Revisted - We Can Still Learn from Polanyi. *Electronic Journal of Knowledge Management*, 5(2) pp. 173-180 (2007)
- Grudin, J.: Groupware and social dynamics: eight challenges for developers. *Commun. ACM*, 37(1) pp. 92-105 (1994)
- Grudin, J.: Evaluating Opportunities for Design Capture. *Design Rationale Concepts, Techniques, and Use*. Lawrence Erlbaum Associates, Mahwah, NJ (1996) pp. 453-470
- Heer, J., Card, S.K. and Landay, J.A.: Prefuse: a toolkit for interactive information visualization. In: Proc. SIGCHI conference on Human factors in computing systems, pp. 421-430 (2005)
- Holten, D., Cornelissen, B. and van Wijk, J.J.: Trace Visualization Using Hierarchical Edge Bundles and Massive Sequence Views. In: Maletic, J.I., Telea, A. and Marcus, A. (eds.): In: Proc. Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007. 4th IEEE International Workshop on, pp. 47-54, Banff, Canada (2007)
- Jansen, A. and Bosch, J.: Evaluation of tool support for architectural evolution. In: Proc. Automated Software Engineering, 2004. Proceedings. 19th International Conference on, pp. 375-378 (2004)
- Jansen, A. and Bosch, J.: Software Architecture as a Set of Architectural Design Decisions. In: Proc. Fifth Working IEEE/IFIP Conference on Software Architecture (WICSA 2005), pp. 109-120. IEEE Computer Society, Pittsburgh, PA, USA (2005)
- Jansen, A., Van der Ven, J., Avgeriou, P. and Hammer, D.: Tool support for architectural decisions. In: Proc. Sixth Working IEEE/IFIP Conference on Software Architecture (WICSA 2007), Mumbai (2006)
- Karsenty, L.: An empirical evaluation of design rationale documents. In: Tauber, M.J. (ed.): In: Proc. SIGCHI Conference on Human Factors in Computing Systems: Common Ground (CHI '96), pp. 150-156. ACM Press, New York, NY (1996)
- Kienle, H.M. and Müller, H.A.: Requirements of Software Visualization Tools: A Literature Survey. In: Proc. Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007. 4th IEEE International Workshop on, pp. 2-9 (2007)
- Klein, M.: Capturing design rationale in concurrent engineering teams. *IEEE Computer*, 26(1) pp. 39-47 (1993)
- Kruchten, P.: An Ontology of Architectural Design Decisions. In: Bosch, J. (ed.): In: Proc. 2nd Groningen Workshop on Software Variability Management, pp. 55-62. Rijksuniversiteit Groningen, Groningen, NL (2004)
- Kruchten, P., Lago, P. and van Vliet, H.: Building up and reasoning about architectural knowledge. In: Hofmeister, C. (ed.): QoSA-Quality of Software Architecture, vol. 4214. Springer-Verlag, Vaesteras, Sweden (2006) pp. 43-58
- Kruchten, P., Lago, P., van Vliet, H. and Wolf, T.: Building up and exploiting architectural knowledge. In: Proc. Working IEEE/IFIP Conference on Software Architecture (WICSA) 2005, pp. 291 - 292. IEEE Computer Society, Pittsburgh, PA (2005)
- Kunz, W. and Rittel, H.W.J.: Issues as Elements of Information Systems, Working Paper 131. The University of California at Berkeley (1970)
- Lago, P. and van Vliet, H.: Explicit Assumptions Enrich Architectural Models. In: Proc. International Conference on Software Engineering (ICSE 2005), pp. 206-214. ACM Press, St. Louis, MO, USA (2005)

- Lee, J.: SIBYL: a tool for managing group design rationale. In: Proc. ACM conference on Computer-supported cooperative work (CSCW90), pp. 79 - 92, Los Angeles (1990)
- Lee, J.: Extending the Potts and Bruns model for recording design rationale. In: Proc. Software Engineering, 1991. Proceedings., 13th International Conference on, pp. 114 - 125 (1991)
- Lee, J.: Design Rationale Systems: Understanding the Issues. IEEE Expert, 12(3) pp. 78-85 (1997)
- Lee, J. and Lai, K.-Y.: What's in Design Rationale? Design Rationale: Concepts, Techniques, and Use. Lawrence Erlbaum Associates, Inc., Mahwah, NJ (1996) pp. 21-51
- Lee, L. and Kruchten, P.: Capturing Software Architectural Design Decisions. In: Proc. 20th Canadian Conference on Electrical and Computer Engineering, pp. 686-689. IEEE, Vancouver, BC, Canada (2007)
- Lee, L. and Kruchten, P.: Customizing the Capture of Software Architectural Design Decisions. In: Proc. 21st Canadian Conference on Electrical and Computer Engineering (CCECE 2008), pp. 693-698. IEEE, Niagara Falls, ON, Canada (2008a)
- Lee, L. and Kruchten, P.: A Tool to Visualize Architectural Design Decisions. In: Becker, S. and Plasil, F. (eds.): In: Proc. Fourth International Conference on the Quality of Software Architectures (QoSA 2008), vol. LNCS 5281, pp. 43-54. Springer, Heidelberg, Karlsruhe, Germany (2008b)
- Lee, L. and Kruchten, P.: Visualizing Software Architectural Design Decisions. In: Morrison, R., Balasubramaniam, D. and Falkner, K. (eds.): In: Proc. Second European Conference on Software Architecture (ECSA 2008), vol. LNCS 5292, pp. 359-362. Springer-Verlag, Paphos, Cyprus (2008c)
- MacLean, A., Young, R.M., Bellotti, V.M.E. and Moran, T.P.: Questions, options, and criteria: Elements of design space analysis. Human-Computer Interaction, 6 pp. 201-250 (1991)
- Moreta, S. and Telea, A.: Visualizing Dynamic Memory Allocations. In: Maletic, J.I., Telea, A. and Marcus, A. (eds.): In: Proc. Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007. 4th IEEE International Workshop on, pp. 31-38, Banff, Canada (2007)
- Munzner, T.: Interactive Visualization of Large Graphs and Networks. Ph.D. Dissertation. Department of Computer Science, Stanford University (2000)
- Nonaka, I.: The knowledge-creating company. Harvard Business Review, vol. 69 (1991) pp. 96-104
- Polanyi, M.: The Tacit Dimension. Routledge & Kegan Paul, London (1966)
- Potts, C. and Bruns, G.: Recording the reasons for design decisions. In: Proc., pp. 418-427. IEEE Computer Society, Singapore (1988)
- Regli, W.C., Hu, X., Atwood, M. and Sun, W.: A survey of design rationale systems: Approaches, representation, capture and retrieval. Engineering with Computers, 16(3-4) pp. 209-235 (2000)
- Robillard, P.N.: The role of knowledge in software development. Commun. ACM, 42(1) pp. 87-92 (1999)
- Sawant, A.P. and Bali, N.: SoftArchViz: A Software Architectural Visualization Tool. In: Maletic, J.I., Telea, A. and Marcus, A. (eds.): In: Proc. Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007. 4th IEEE International Workshop on, pp. 154-155, Banff, Canada (2007)

- Schuster, N.: ADkwik – a Collaborative System for Architectural Decision Modeling and Decision Process Support based on Web 2.0 Technologies. Diplomarbeit im Studiengang Medieninformatik Doctoral Thesis. Studiengang Medieninformatik, Hochschule der Medien (2007)
- Selvin, A.M., Buckingham Shum, S., Sierhuis, M., Conklin, J., Zimmermann, B., Palus, C., Drath, W., Horth, D., Domingue, J., Motta, E. and Li, G.: Compendium: Making Meetings into Knowledge Events. In: Proc. Knowledge Technologies, Austin, TX (2001)
- Sierhuis, M. and Selvin, A.M.: Towards a Framework for Collaborative Modeling and Simulation. In: Proc. Workshop on Strategies for Collaborative Modeling and Simulation Conference on Computer-Supported Collaborative Work (CSCW '96), pp. 1-7, Boston, MA (1996)
- Storey, M.-A.D., Cheng, L.T., Bull, R.I. and Rigby, P.C.: Waypointing and Social Tagging to Support Program Navigation. CHI '06 extended abstracts on Human factors in computing systems. ACM Press, Montréal, Québec (2006) pp. 1367-1372
- Tang, A., Babar, M.A., Gorton, I. and Han, J.: A Survey of Architecture Design Rationale. Journal of Systems and Software, 79(12) pp. 1792-1804 (2006)
- Ting, E.: Design Decision Tools Experiment. Unpublished M.Eng. Project Report, University of British Columbia (2009)
- Tyree, J. and Ackerman, A.: Architecture Decisions: Demystifying Architecture. IEEE Software, 22(2) pp. 19-27 (2005)
- van der Ven, J.S., Jansen, A.G.J., Avgeriou, P. and Hammer, D.K.: Using Architectural Decisions. In: Proc. 2nd International Conference on the Quality of Software Architectures (QoSA 2006), pp. 1-10, Västerås, Sweden (2006)
- van Gurp, J. and Bosch, J.: Design erosion: problems and causes. Journal of Systems and Software, 61(2) pp. 105-119 (2002)
- Wettel, R. and Lanza, M.: Visualizing Software Systems as Cities. In: Maletic, J.I., Telea, A. and Marcus, A. (eds.): In: Proc. Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007. 4th IEEE International Workshop on, pp. 92-99, Banff, Canada (2007)
- Wild, C. and Maly, K.: Towards a software maintenance support environment. In: Proc. Proceedings of the Conference on Software Maintenance, pp. 80-85 (1988)
- Wild, C., Maly, K., Liu, L., Chen, J.-S. and Xu, T.: Decision-based software development: design and maintenance. In: Proc. Conference on Software Maintenance, 1989, p. 297 (1989)
- Wolf, T. and Dutoit, A.H.: Sysiphus: Combining system modeling with collaboration and rationale. Softwaretechnik-Trends, 24(4) (2004)
- Zimmermann, B. and Selvin, A.M.: A framework for assessing group memory approaches for software design projects. In: Proc. 2nd conference on Designing interactive systems: processes, practices, methods, and techniques, pp. 417-426. ACM, Amsterdam, The Netherlands (1997)
- Zimmermann, O., Koehler, J. and Leymann, F.: Architectural Decision Models as Micro-Methodology for Service-Oriented Analysis and Design. In: Lübke, D. (ed.): In: Proc. Workshop on Software Engineering Methods for Service-oriented Architecture 2007 (SEMSOA 2007), vol. 244, pp. 46-60. CEUR-WS.org, Hannover, Germany (2007)

APPENDIX A – ADDEX USER’S GUIDE

User’s Guide

The end-user’s introduction to the ADDEX tool is included in this appendix section.

ADDEX User's Guide

*End User's Introduction to the
Architectural Design Decision Exploration (ADDEX) Tool*

Larix Lee

November 20, 2008

Version 1.0

The tool described in this user's guide is created as a part of the author's academic Master's research thesis.

Table of Contents

Purpose	3
Getting Started	3
System Requirements.....	3
Installing and Starting ADDEX	4
The ADDEX Tool Overview	5
Using ADDEX.....	5
Decision Capture.....	5
Formal elicitation – DecisionCaptureTool.....	6
Lightweight top-down decision capture – DecisionStickies	7
Lightweight bottom-up decision capture – DecisionCapturePlugin	9
Decision Exploration.....	11
Tabular Listing.....	11
Decision Structure Visualization	12
Decision Chronology Visualization	14
Decision Impact Visualization.....	15
Reference: Design Decision Structures.....	17

Purpose

This document is intended to provide a general overview of the Architectural Design Decision Exploration (ADDEX) tool to the end-users of this tool. The document begins by outlining the basic structure of the tool and then it describes step-by-step procedures for basic tool functionality. By the end of this document, the end-users reading this document should be able to understand how to use the ADDEX tool and get started on tasks related to architectural design decision exploration and analysis.

Getting Started

The first step to use the ADDEX tool is to install the tool. The tool runs on any computer system platform that supports the Java 5 runtime environment. After installation, ADDEX can be started simply by opening the Java jar file.

System Requirements

The recommended system requirements to install and use the ADDEX tool are:

- Java VM 1.5- supported personal computer system
- Java 5 or greater runtime environment
- Graphical user environment and display
- 256 MB available system RAM
- Pentium II-class, G3 PowerPC or newer system processor
- 30 MB free hard drive or storage space
- Keyboard and two-button mouse

Optional installed prerequisites for additional functionality:

- Network-capable system
- MySQL 5.0 database server software
- Eclipse 3.2 or higher integrated development environment (IDE)

Installing and Starting ADDEX

There are three binary files that make up the ADDEX tool:

- ca.ubc.ece.seal.ADDEX.DecisionCapturePlugin.1.0.2.jar
- ADDEX.DecisionStickies.1.0.2.jar
- ADDEX.DecisionExploration.1.0.2.jar

The Java 5 Runtime Environment must be already installed and configured on the target computer system. The Eclipse IDE should be installed in order to install and use the optional decision capture Eclipse plug-in.

Installation of the tool is performed by copying three Java Jar files in two steps:

- 1) The Eclipse plug-in decision capture component can be installed by copying the ca.ubc.ece.seal.ADDEX.DecisionCapturePlugin.1.0.2.jar file to the "plugin" folder in the Eclipse installation folder.
- 2) Copy the remaining two Jar files to an easily-accessible folder with read/write permissions. Installation is not required but is highly recommended.

To use the plug-in, start Eclipse as normal. To use the DecisionStickies decision capture component, double-click the ADDEX.DecisionStickies.1.0.2.jar file to open it. To use the formal elicitation/visualization components, double-click the ADDEX.DecisionExploration.1.0.2.jar file to open it.

In future releases, an installer will be used to automate the installation process and a launcher application will be used to start all four components.

Tip: Some systems and/or configurations cannot execute Java Jar files by double-clicking the file. In those cases, the ADDEX tool components can be started using the Java Runtime command-line interface. At a command terminal, navigate to the folder containing the two Java Jar files and enter the respective command listed below.

To run the DecisionStickies component:

```
java -jar ADDEX.DecisionStickies.1.0.2.jar
```

To run the formal elicitation/visualization components:

```
java -jar ADDEX.DecisionExploration.1.0.2.jar
```

The ADDEX Tool Overview

The ADDEX tool is made up of four smaller tools (components) that share a common decision representation, storage and manipulation framework to address the capture and exploration of software architectural design decisions. These four components are: 1) formal elicitation; 2) lightweight top-down capture; 3) lightweight bottom-up capture; and 4) decision visualization.

Figure 6 below illustrates the system structure of the ADDEX tool. The lightweight top-down capture component in the ADDEX tool is also known as the “DecisionStickies” tool. The lightweight bottom-up capture component is an Eclipse Plug-in. Formal elicitation and decision visualization components are integrated into a single package (generalized as “DecisionExploration”). Within the decision visualization component, four visualization aspects are found. The goal of these four visualization aspects is to support decision exploration by visualizing the different facets of the architectural knowledge found within the captured design decisions.

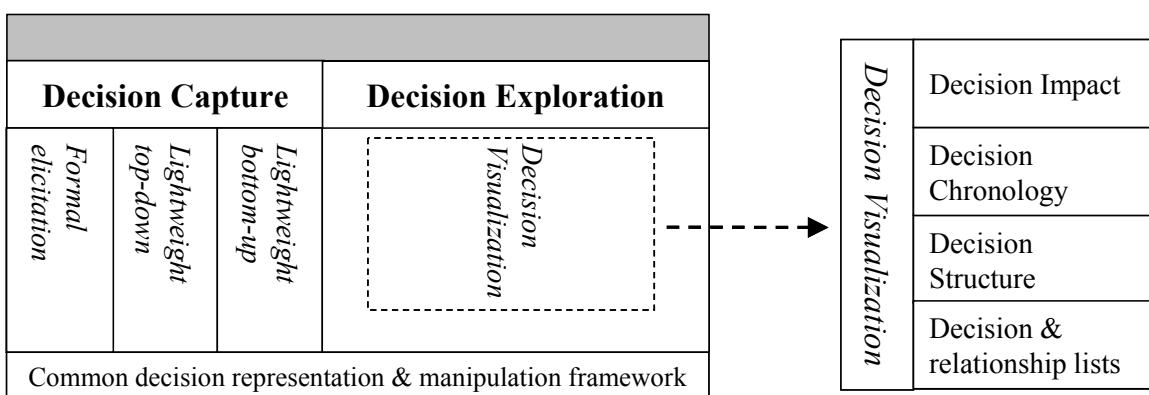


Figure 1: ADDEX system diagram. Four smaller tools (components) make up the ADDEX tool and are tied together through a common framework for decision representation, storage, and manipulation. The visualization tool contains four distinct visualization aspects that can be used to explore architectural design decisions.

Using ADDEX

The ADDEX tool has four components. Three of the ADDEX components correspond to the capture of architectural design decisions. The fourth component deals with architectural design decision exploration. This section will describe how to use the ADDEX components for decision capture and exploration.

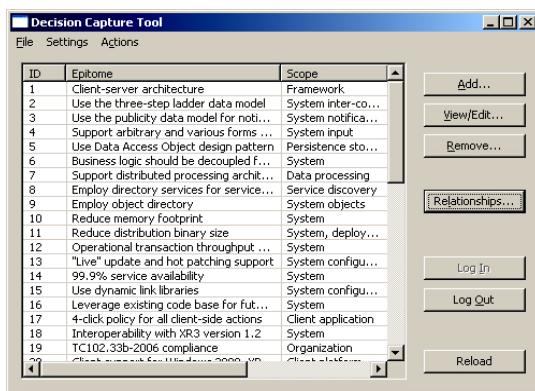
Decision Capture

The three ADDEX components involving decision capture are: Formal elicitation, lightweight top-down (DecisionStickies), and lightweight bottom-up (DecisionCapturePlugin). For details on how to install and start the respective tools, refer to the *Getting Started* section.

Formal elicitation – DecisionCaptureTool

Starting DecisionCaptureTool

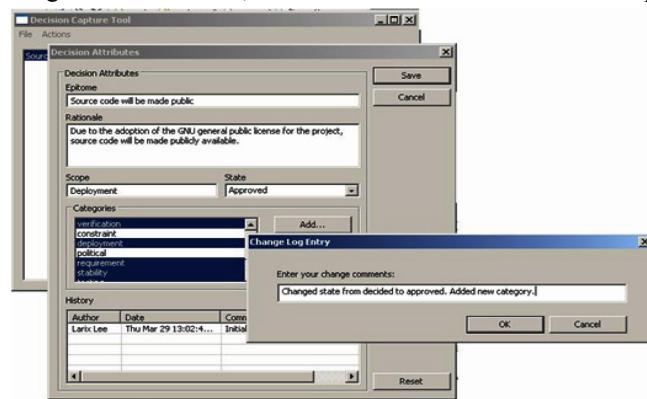
- 1) The formal elicitation component of the ADDEX tool can be started by double-clicking on the Java Jar file named ADDEX.DecisionExploration.1.0.2.jar.
- 2) Once the application has started, go to the “File” menu and select “open”.
- 3) Enter the name of the current project and click OK.
- 4) A list of previously-created, structured decisions (from the two lightweight capturing components or previous formal elicitation sessions) is displayed and can be viewed and browsed.



- 5) To add, remove, or change the structured decisions, click the “Log in” button. Logging in is required for any changes to the list of design decisions.
- 6) Enter your name and click OK to complete the log in process.

Using DecisionCaptureTool

- **Decision Forming:** To form a decision, click the “add” button under the decisions button group. A dialog box will appear where you may enter specific decision information. Enter the decision information, such as the decision epitome (key idea), the rationale behind the decision, the scope and state. Select a decision state. When done, click the “save” button. A dialog appears where you can enter a change log entry for the modified (new) decision. After entering the change log comment and clicking the “OK” button, the decision is added to the list of captured decisions.



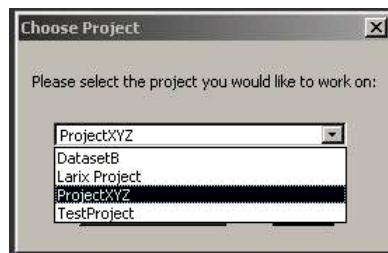
Lightweight top-down decision capture – DecisionStickies

Starting DecisionStickies

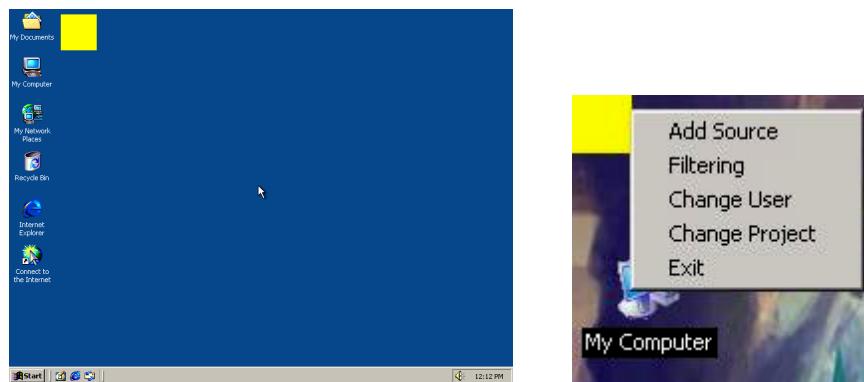
- 1) The first step to capturing decisions using DecisionStickies is to start up the program. You can do so by double-clicking on the Java Jar file named ADDEX.DecisionStickies.1.0.2.jar.
- 2) Next, log in to the ADDEX tool by entering your name and your chosen password. If you don't have a configured name/password, you can create a new user by clicking on the "create new user" button.



- 3) Select your project from the drop down list and click "done". If your project is not listed, you can create a new project by clicking "create a new project".

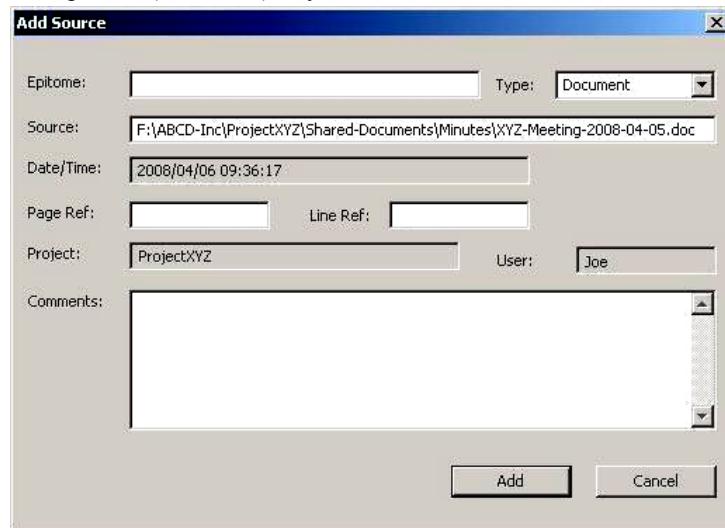


- 4) The main user interface is presented to you as a yellow square on the upper left corner of your screen. Right-clicking on the decision yellow square will enable you to access other features.

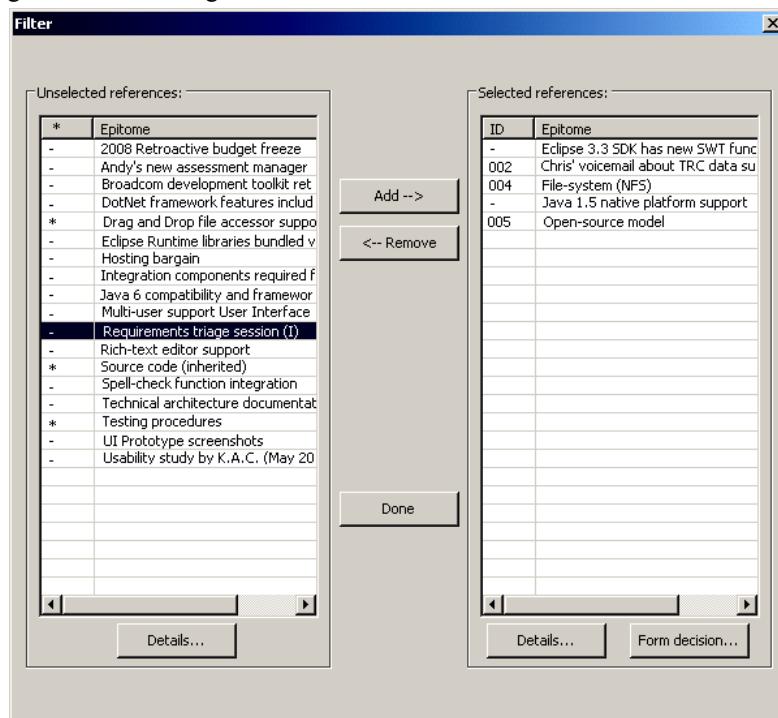


Using DecisionStickies

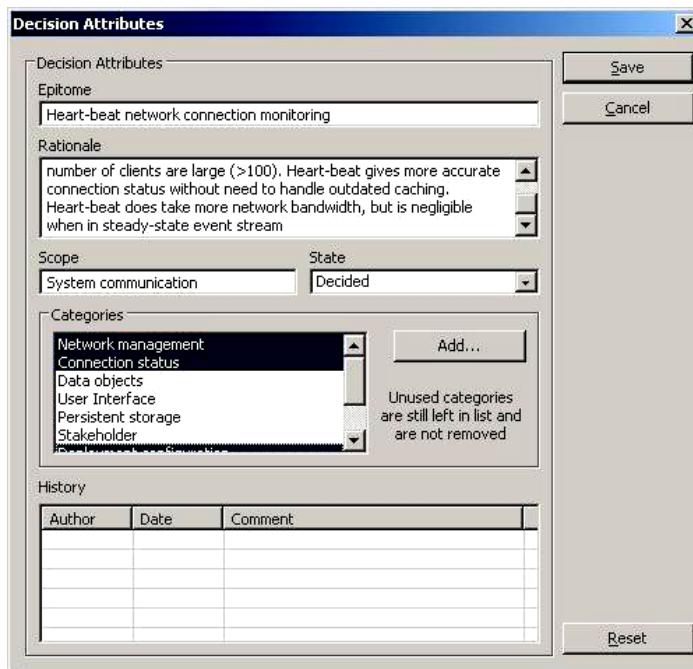
- 1) **Decision flagging:** Decisions can be flagged as a decision by drag-and-dropping the document on top of the yellow square. A dialog window will pop up, with some fields pre-filled for you, and you can enter the epitome (main idea) of your additional decision information.



- 2) **Decision filtering:** When you have many decisions, you can filter your decisions for relevance by right-clicking on the yellow square and selecting “filtering”. A dialog screen appears and shows two lists (all decisions and selected decisions). Select a decision by moving a decision from the left list (all decisions) to the right list (selected decision) by clicking on the decision reference on the left list and clicking “add”. Similarly, you can remove a selected decision by selecting a decision from the right list and clicking “remove”.



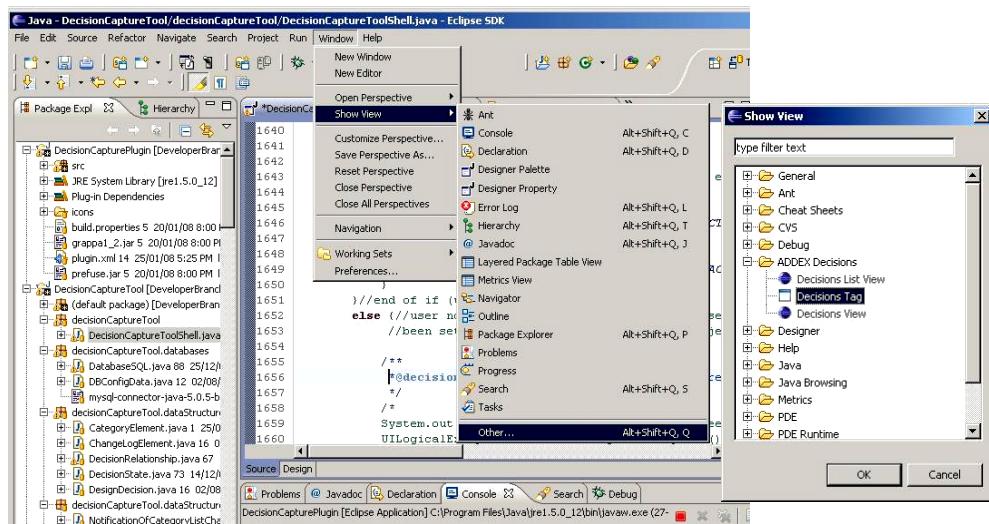
- 3) **Decision forming:** For the selected decisions you can structure the reference formally by clicking on the selected decision references in the filtering dialog and then clicking on the “form decision” button. A dialog window will show up with several more fields to complete the decision structure. Some of the fields are already pre-filled with the information from the decision reference. Fill out the information pertaining to the decision and click “save” when done to save the structured decision. This decision is now saved to the decision repository to be shared with other users.



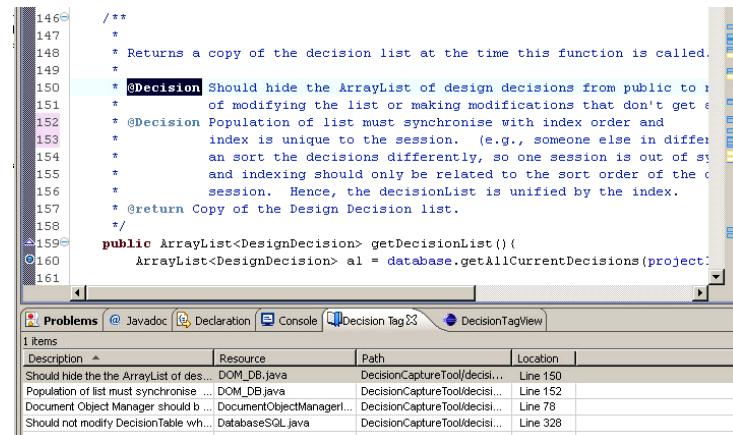
Lightweight bottom-up decision capture – DecisionCapturePlugin

Starting DecisionCapturePlugin

- 1) Once the DeicsionCapturePlugin has been installed, starting the plugin is as simple as starting Eclipse as normal. The main interface of the lightweight bottom-up decision capture is the “Decision Tags” view in Eclipse.
- 2) If the “Decision Tags” view is not visible in when Eclipse is in the opened state, you can open it by going to the “Window” menu, then “show view” submenu, and then selecting “Other...”. A dialog will appear where you can select “Decision Tags” from the list of views. Click OK when done.



- 3) When started, the “Decision Tags” view will scan through all the source code in the active projects and find decision tags (denoted by the //Decision or @Decision prefixes) and lists them in the view at the bottom of Eclipse.



Using DecisionCapturePlugin

- 1) **Decision tagging:** To add a decision tag within Eclipse, browse to the class or function headers of the areas where the decision affects. Insert a code comment beginning with //Decision or @Decision, where the code comment summarizes the key ideas of the decision. Save the modified source files like normal.
- 2) **Decision forming:** To form a decision from the decision tag, perform code check-in. During the code check-in, the source code files are scanned for decision tags. The newly added decision tag is detected and you are shown a dialog to form the decisions. Fill out the additional information pertaining to the decision and click “save”. Continue code check-in as normal.

Decision Exploration

The fourth ADDEX component is decision visualization. As the visualization is designed to support the exploration of design decisions, it is logical to link the visualization component to the formal elicitation component since the structured decisions make information retrieval and analysis easier. Visualization is helpful to make sense of various attributes among a large set of decisions. Therefore, the four visualization aspects (tabular listing, decision structure visualization, decision chronology visualization and decision impact visualization) are found within the formal elicitation component. Many typical decision information manipulation features are found in the tabular listing view. All views update each other when decisions in the decision list are loaded, added, edited, or removed.

Below is how you can start the decision visualization component. As the visualization component is part of the formal elicitation component, the steps to start are similar to steps for the formal elicitation component. For details on how to install and start the respective tools, refer to the *Getting Started* section.

- 1) The formal elicitation component of the ADDEX tool can be started by double-clicking on the Java Jar file named `ADDEX.DecisionExploration.1.0.2.jar`.
- 2) The first screen shown after starting DecisionExploration is the Tabular Listing aspect.
- 3) The other three aspects can be opened by selecting the respective visualization aspects through the “visualization” menu. All four visualization aspects can be opened and viewed simultaneously.

Tabular Listing

- **Decision browsing:** Decisions (and relationships) are displayed visually using tables. Each decision is represented as a row in a table. The columns in the table display the decision attributes.

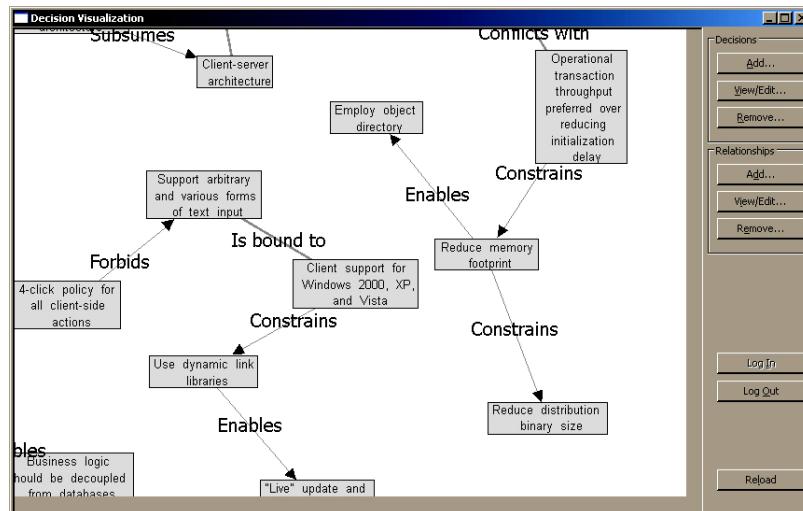
The screenshot shows the 'Decision Capture Tool' interface. On the left is a window titled 'Decision Relationships' containing a table of decision relationships. The columns are 'Source decision', 'Relationship', 'Target decision', and 'Bi-directional'. The table lists various relationships between decisions like 'Subsumes', 'Enables', 'Conflicts', etc. On the right is a larger window showing a list of decisions in a table with columns 'ID', 'Epitome', and 'Scope'. The decisions listed include 'Client-server architecture', 'Use the three-step ladder data model', 'Support arbitrary and various forms...', 'Use Data Access Object design pattern', 'Business logic should be decoupled...', 'Support distributed processing arch...', 'Employ directory services for service...', 'Employ object directory', 'Reduce memory footprint', 'Reduce distribution binary size', 'Operational transaction throughput...', '"Live" update and hot patching support', '99.9% service availability', 'Use dynamic link libraries', 'Leverage existing code base for fut...', '4-click policy for all client-side actions', 'Interoperability with XR3 version 1.2', and 'TC102.33b-2006 compliance'. Below the main table are buttons for 'Add...', 'View/Edit...', and 'Remove...'. A 'Relationships' button is also visible at the top of the main window.

- **Adding decisions:** Decisions are added using the formal elicitation decision capturing approach. The details on adding decisions are described in the “Formal Elicitation” section above. Decision creation requires a user to be logged in.

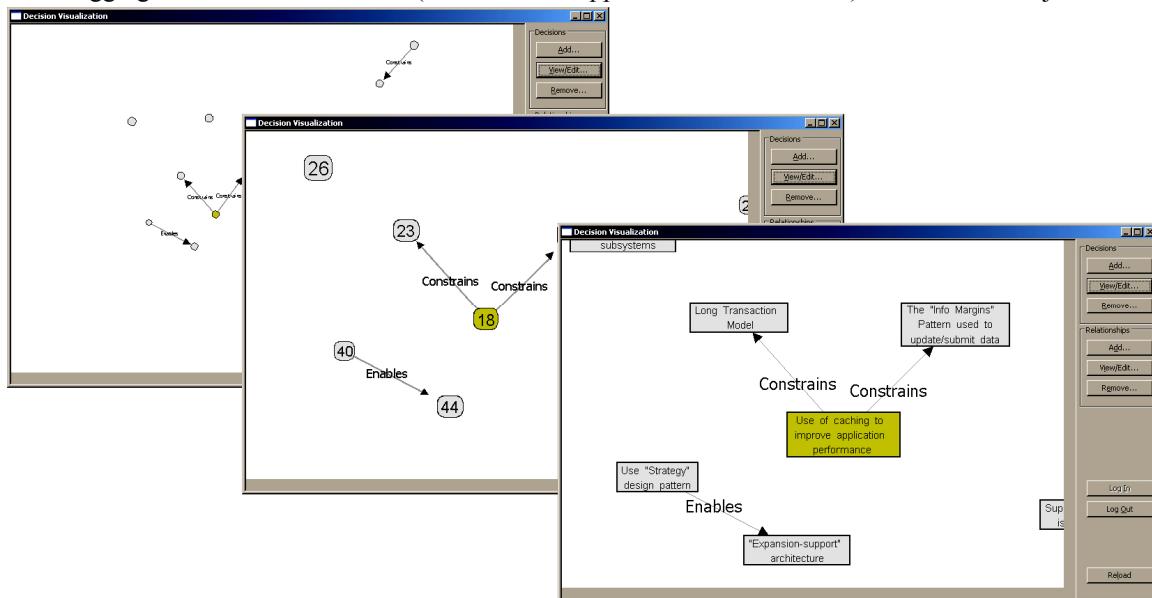
- **Viewing/editing/removing decisions:** Decisions can be viewed or edited by selecting the decision of interest and clicking the “view/edit” button. In order to edit the decisions, you must log in first. Decisions can be removed by selecting the decision and clicking the “remove” button. Note, decisions are removed from view but remains in the decision list for documentation purposes. It is suggested to use the “obsolete” or “rejected” decision states instead of removing decisions.
- **Viewing decision history:** Each time you edit and save the decision, the old decision information is kept and stored in a history. You can view the decision history by viewing a decision and looking at the history table at the bottom of the decision dialog. Double-clicking on a history row item would bring up another decision dialog with the decision information specific to that version.
- **Adding/viewing/removing decision relationships:** Decisions can be related to each another. To add or remove decision relationships, click the “relationships” button. A new dialog with a list of relationships appears. Relationship details can be viewed using the list or by selecting a relationship and then clicking “view”. Click the “remove” button to remove a relationship. Click the “add” button to bring up a dialog where you can select the two decisions and the relationship type.
- **Saving and retrieving a list of decisions:** A list of decisions can be saved by going to the “File” menu, selecting “save”, typing in the name of the project, and clicking “OK” to save. The saved file will be stored in the same directory as the application. To open the list of decisions, go to the “File” menu and selecting “open”. Enter the name of the project and select “OK”.
- **Importing and exporting a list of decisions using XML:** Decisions can be exported by selecting “import” from the “File” menu, entering the path and file name of the XML file to export, then clicking “OK”. To import an XML file, select “import...” from the “File” menu, select the path and file of XML file to import and click “OK”.
- **Database connectivity:** To use an existing SQL database server to store and retrieve decisions, go to the options” menu and select “database”. Fill in the necessary database server information and click OK. The ADDEX tool will create the database for you if it does not exist on the database server. Decisions are saved and retrieved using the database automatically when there is a database configuration set for the ADDEX tool.

Decision Structure Visualization

- **Decision browsing:** Decisions are displayed visually using graphs. Decisions are nodes and relationships are edges. They are arranged using an animated force-directed layout. More mature decisions (decisions in the “decided” or “approved” states) are rendered using larger nodes than less mature ones (like “idea”, “tentative”, or “rejected”).



- Interacting with the set of decisions:** *Semantic zooming* allows more decision information to be displayed in the nodes when zoomed in on a set of decisions. Less information will be displayed when zoomed out. Zooming in is performed by centering the mouse cursor to where you want to zoom in, holding the right-mouse-button and moving the mouse forward. To zoom out, hold the right-mouse-button and move the mouse backwards. Right-clicking anywhere on the visualization will reset the zoom. Decisions can be dragged around and other decisions will react to the dragging. More mature decisions ("decided" or "approved" decision states) act as heavier objects.

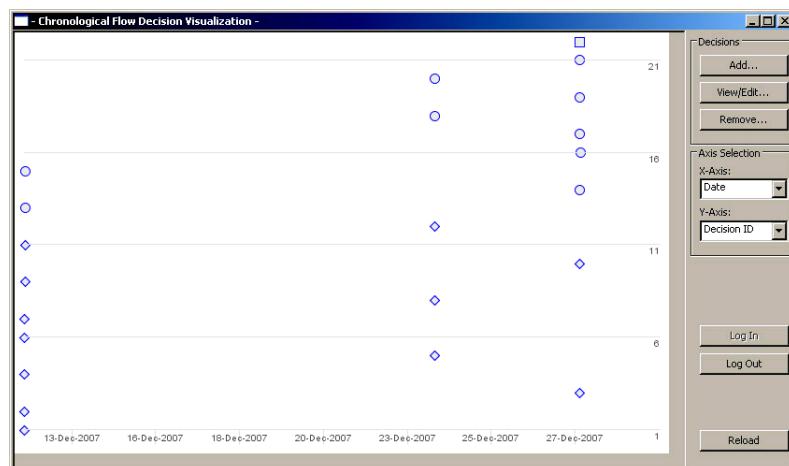


- Adding/removing decisions:** Decisions can be added by clicking on the "add decision" button in on the right side of the visualization. Decisions are added in a way similar to the method in Tabular Listing. The added decision will appear as a new node in the visualization. Remove a decision by selecting the decision in the visualization and clicking "remove". All relationships associated to this decision will be removed.

- **Viewing/editing decisions:** Decisions can be viewed or edited by selecting (clicking) the decision of interest in the visualization and then clicking the “view/edit” button. In order to edit the decisions, you must log in first.
- **Viewing decision history:** When viewing a decision, you can view the history of a decision by double-clicking on a row item in the history table at the bottom of the decision dialog. Another decision dialog appears with the decision information specific to that version.
- **Adding/viewing/removing decision relationships:** To view a relationship, select a relationship (line that connects two decisions together) in the visualization and then click the “view relationships” button. To add a decision relationship, click the “add relationship” button. A dialog appears where you can select two decisions and the relationship type between them. To remove a decision, select the relationship in the visualization and click “remove”.

Decision Chronology Visualization

- **Decision browsing:** Decisions are displayed visually in a timeline. Decisions are nodes and the horizontal axis (x-axis) denotes time. The flow of time goes from left to right – from earliest to most recent. The shape (size) of the nodes denotes the decision states. The vertical axis (y-axis) represents a user-selected decision attribute. For example, the y-axis can be sorted by decision ID to identify decision changes, or the y-axis can be sorted by author to identify critical and subversive stakeholders.

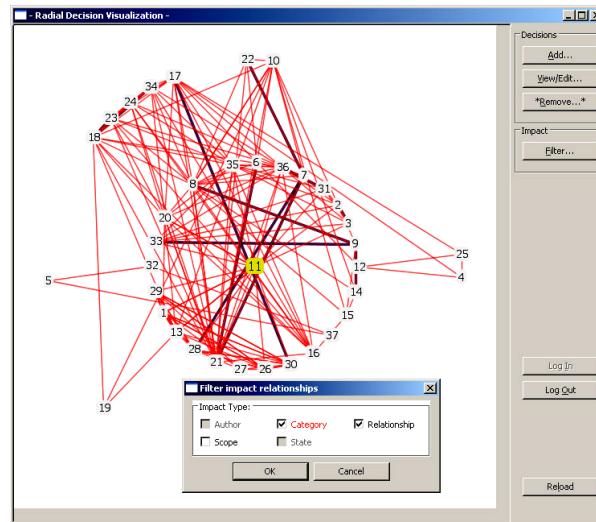


- **Interacting with the set of decisions:** You can zoom in on the timeline by click-and-dragging around a set of closely-spaced decisions to reduce the time range and effectively spread out the decisions visually. Right-clicking anywhere on the visualization will reset the zoom.
- **Adding decisions:** Decisions can be added by clicking on the “add decision” button in on the right side of the visualization. Decisions are added in a way similar to the method in Tabular Listing. The added decision will appear as a new node in the visualization and, if necessary, the timeline range will be updated to reflect this update.

- **Viewing/editing decisions:** Decisions can be viewed or edited by selecting (clicking) the decision of interest in the visualization and then clicking the “view/edit” button. In order to edit the decisions, you must log in first.
- **Viewing decision history:** When viewing a decision, you can view the history of a decision by double-clicking on a row item in the history table at the bottom of the decision dialog. Another decision dialog appears with the decision information specific to that version.

Decision Impact Visualization

- **Decision browsing:** Decisions are displayed visually using a radial graph layout. Decisions are nodes and the *impact*-relationships are the edges. Decisions are positioned concentrically around a decision in the centre. The node in the centre is the decision of interest. Decisions in the immediate concentric circle of decisions surrounding the centre decision are decisions that are directly impacted by that decision. The outer concentric decision circles surrounding the decision of interest are decisions that are indirectly impacted by the decision of interest. The further out from the centre decision the less direct the impact.



- **Interacting with the set of decisions:** Clicking on any decision makes the selected decision the decision of interest, where the animated layout will reorganize the decisions around the new decision. The visualization supports zooming. You can zoom in on the set of decisions by centering the mouse cursor to where you want to zoom in, holding the right-mouse-button and moving the mouse forward. To zoom out, hold the right-mouse-button and move the mouse backwards. Right-clicking anywhere on the visualization will reset the zoom.
- **Adding decisions:** Decisions can be added by clicking on the “add decision” button in on the right side of the visualization. Decisions are added in a way similar to the method in Tabular Listing. The added decision will be analyzed for decision impact and will appear as a new node in the visualization.

- **Viewing/editing decisions:** Decisions can be viewed or edited by selecting (clicking) the decision of interest in the visualization and then clicking the “view/edit” button. In order to edit the decisions, you must log in first.
- **Viewing decision history:** When viewing a decision, you can view the history of a decision by double-clicking on a row item in the history table at the bottom of the decision dialog. Another decision dialog appears with the decision information specific to that version.
- **Filtering the decision impact relationships:** Decision impact relationships can be added or removed from view by selecting the “filter” button on the screen.



Reference: Design Decision Structures

This section describes the architectural design decision representation model used by the ADDEX tool. Each architectural design decision has certain attributes to describe the decision. These attributes and how they are represented are summarized in Table 1.

Table 1: Attributes of decisions

Name	Type
Epitome	Text
Rationale	Text or pointer
Scope	Text
State	Enumeration
History	List of (time stamp + author + change)
Categories	List
Publicity Level	Enumeration
Source (or expert)	Text

The epitome describes the essence of the decision and is supported by reasons stated in the rationale; however, the decision context is restricted by the scope of the decision. Each decision has a certain state, which describes the “maturity” of the decision. The states and its transition paths are depicted in figure 2 below. Any change made to the decision attributes are logged in the decision history. The category attribute complements the decisions with additional information. The publicity level attribute sets the level of decision disclosure for the selective-release of design decisions, while the source/expert attribute can document where the knowledge is found for traceability or to support decisions intentionally left tacit.

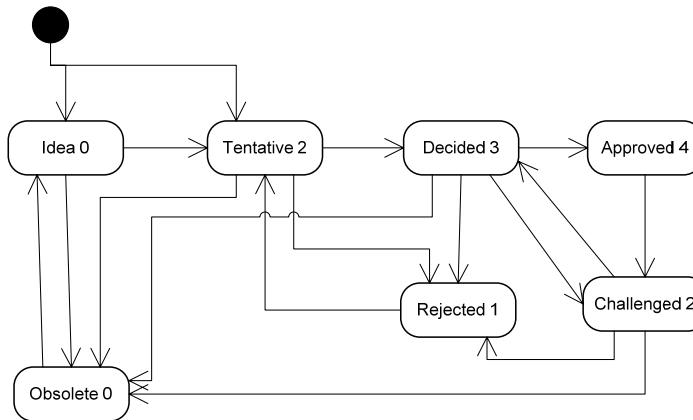


Figure 2: UML state diagram of decision states and their transitions. The number next to each state name is the promotion level for each state. Higher numbers mean greater levels, implying a higher decision “weight”. Arrows leading out from a state denote the transition paths for that decision state. Created decisions start out in the “idea” or “tentative” states. Decisions are never removed; they are given a new state (“rejected” or “obsolete”).

There are ten inter-decision relationships. Table 2 shows the ten relationship classifications between decisions, and these relationships are of the form, “Decision A ‘is related to’ Decision B”.

Table 2: Decision relationships

Relationship Type	Association
Constrains	Directional
Forbids	Directional
Enables	Weak directional
Subsumes	Directional
Conflicts with	Bidirectional
Overrides	Directional
Comprises (is made of)	Directional
Is bound to	Strong bidirectional
Is an alternative to	Directional
Is related to	Weak directional

Decisions can constrain one another, where the affected decision is contingent to the constraining decision. The weak form of this relationship is known as the enabling relationship, while the bi-directional form is strong and is known as the binding relationship. Decisions could also forbid another decision from being made, or could be more encompassing than another (subsumes). Decision conflicts are symmetrical and are possible when both decisions are mutually exclusive and have the same scope. Although similar in description, alternatives differ from conflict relationships. Alternatives are decisions that address the same issue and scope, but can be replaced by one or another, which relates various choices together. Neither alternatives nor conflicts are subsets of each other. Decisions could also override one another, or can break down into other decisions or comprises. If a decision relationship does not fit into any of the above types, then the relating relationship can be used, but this is a weak relationship and is used primarily for documentation and illustrative reasons. The implication of relationships is that the decisions can now tell a story of the design process, bringing decision hierarchy and structure to the captured architectural knowledge.

APPENDIX B – ETHICS APPROVAL

Certificate of Approval

The UBC Research Ethics Board Certificate of Approval and its renewal certificate are included in this appendix section.



The University of British Columbia
Office of Research Services
Behavioural Research Ethics Board
Suite 102, 6190 Agronomy Road, Vancouver,
B.C. V6T 1Z3

CERTIFICATE OF APPROVAL - FULL BOARD

PRINCIPAL INVESTIGATOR: Philippe Kruchten	INSTITUTION / DEPARTMENT: UBC/Applied Science/Electrical and Computer Engineering	UBC BREB NUMBER: H07-01139
INSTITUTION(S) WHERE RESEARCH WILL BE CARRIED OUT:		
Institution		Site
UBC		Point Grey Site
Other locations where the research will be conducted: The study, gathering of data, and survey will take place at the participant's choice of their office, home, or any locations designated by the user as an area where they have access to a computer and frequently make software decisions.		
CO-INVESTIGATOR(S): Larix Lee		
SPONSORING AGENCIES: Natural Sciences and Engineering Research Council of Canada (NSERC)		
PROJECT TITLE: Capture and Visualization of Software Architecture Design Decisions		
REB MEETING DATE: June 14, 2007	CERTIFICATE EXPIRY DATE: June 14, 2008	
DOCUMENTS INCLUDED IN THIS APPROVAL:		DATE APPROVED: July 12, 2007
Document Name	Version	Date
Consent Forms: Study consent	N/A	June 19, 2007
Advertisements: Call for study participants	N/A	June 19, 2007
Questionnaire, Questionnaire Cover Letter, Tests: Study interview questions	N/A	April 30, 2007
Study survey questionnaire	N/A	April 30, 2007
Letter of Initial Contact: Initial contact letter	N/A	June 19, 2007
The application for ethical review and the document(s) listed above have been reviewed and the procedures were found to be acceptable on ethical grounds for research involving human subjects.		
<i>Approval is issued on behalf of the Behavioural Research Ethics Board and signed electronically by one of the following:</i>		
<hr/> <p style="text-align: center;">Dr. Peter Suedfeld, Chair Dr. Jim Rupert, Associate Chair Dr. Arminee Kazanjian, Associate Chair Dr. M. Judith Lynam, Associate Chair Dr. Laurie Ford, Associate Chair</p>		



The University of British Columbia
Office of Research Services
Behavioural Research Ethics Board
Suite 102, 6190 Agronomy Road,
Vancouver, B.C. V6T 1Z3

CERTIFICATE OF APPROVAL- MINIMAL RISK RENEWAL

PRINCIPAL INVESTIGATOR: Philippe Kruchten	DEPARTMENT: UBC/Applied Science/Electrical and Computer Engineering	UBC BREB NUMBER: H07-01139
INSTITUTION(S) WHERE RESEARCH WILL BE CARRIED OUT:		
Institution UBC	Site Vancouver (excludes UBC Hospital)	
Other locations where the research will be conducted: The study, gathering of data, and survey will take place at the participant's choice of their office, home, or any locations designated by the user as an area where they have access to a computer and frequently make software decisions.		
CO-INVESTIGATOR(S): Larix Lee		
SPONSORING AGENCIES: Natural Sciences and Engineering Research Council of Canada (NSERC)		
PROJECT TITLE: Capture and Visualization of Software Architecture Design Decisions		

EXPIRY DATE OF THIS APPROVAL: April 30, 2009

APPROVAL DATE: April 30, 2008

The Annual Renewal for Study have been reviewed and the procedures were found to be acceptable on ethical grounds for research involving human subjects.

Approval is issued on behalf of the Behavioural Research Ethics Board

Dr. M. Judith Lynam, Chair
Dr. Ken Craig, Chair
Dr. Jim Rupert, Associate Chair
Dr. Laurie Ford, Associate Chair
Dr. Daniel Salhani, Associate Chair
Dr. Anita Ho, Associate Chair

APPENDIX C – LIST OF PUBLICATIONS

Portions of this thesis have been previously published. The author of this thesis wrote the content in the publications with the guidance and editing from the publications' co-author, Philippe Kruchten. Below is a general outline of where the content of these publications may be found.

Chapters 2, 4, 5, and 6 integrate the following four papers:

Lee, L. and Kruchten, P.: Capturing software architectural design decisions. In: Proc. 20th Canadian Conference on Electrical and Computer Engineering (CCECE 2007), pp. 686-689, Vancouver, Canada (2007). With permission of the IEEE.

Lee, L. and Kruchten, P.: Customizing the capture of software architectural design decisions. In: Proc. 21st Canadian Conference on Electrical and Computer Engineering (CCECE 2008), pp. 693-698, Niagara Falls, Canada (2008). With permission of the IEEE.

Lee, L. and Kruchten, P.: A tool to visualize architectural design decisions. In: Becker, S. and Plasil, F. (eds.): Proc. Fourth International Conference on the Quality of Software Architectures (QoSA 2008), LNCS 5281, pp. 43-54, Karlsruhe, Germany (2008). With permission of Springer Science+Business Media.

Lee, L. and Kruchten, P.: Visualizing software architectural design decisions. Morrison, R., Balasubramaniam, D. and Falkner, K. (eds.): Proc. 2nd European Conference on Software Architecture (ECSA 2008), LNCS 5292, pp. 359-362. Paphos, Cyprus (2008). With permission of Springer Science+Business Media.