UNIVERSITY OF CALGARY

A Qualitative Empirical Study of Software Design Decisions made by Designers and Small Teams Cognizant of Agile Practices or Principles

by

Carmen Zannier

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE

DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

CALGARY, ALBERTA

August, 2007

UNIVERSITY OF CALGARY

FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled “A Qualitative Empirical Study of Software Design Decisions made by Designers and Small Teams Cognizant of Agile Practices or Principles” submitted by Carmen Zannier in partial fulfillment of the requirements for the degree of Doctor of Philosophy).

_____

Supervisor, Frank Maurer, Department of Computer Science

_____

Guenther Ruhe, Department of Computer Science

_____

Sheelagh Carpendale, Department of Computer Science

_____

Ron Wardell, Faculty of Environmental Design

_____

External Examiner, Helen Sharp, The Open University

_____

Date

ABSTRACT

This research presents a qualitative empirical study of software design decision making. Decisions are ubiquitous in software design and there are strong calls to examine decision making. However few empirical studies exist that describe how decisions are made in software design environments. This research addresses this limitation. A qualitative empirical investigation is performed consisting of 59 design decision case studies. The investigation employs interviewing, on-looker observations and participatory observations in its data collection. The investigation employs content analysis, explanation building and the development of case study summaries in its data analysis. These techniques are applied across three multi-case studies. Two main conclusions, a set of eight smaller results, and five design decision models are the result. The first conclusion is that software designers appropriate a solution to a design decision approximately as often as they strive for an optimal or boundedly optimal design solution. That is, they apply a solution to a design problem that "just fits" or is "suitable" approximately as often as they look for a good design solution in a pool of potential design solutions. The second conclusion is that the strength and openness of a designer's mental model affects the extent to which alternatives are considered in a design decision. The set of eight smaller results address the effect of time pressure, knowledge acquisition, and agile environments on design decision making, and provide a comparison of existing decision making theories to an empirically-based description of design decision making. The decision models summarize Appropriating, Reasoning and Explaining as 3 types of decision making scenarios addressing small, medium and large decisions. The activities involved in these decision making scenarios are Rationalizing, Implementing, Discussing, Searching, Comparing and Considering. This research impacts the software engineering community in four ways. First it addresses the debate between design as art and design as science. Second it addresses the gap between a top-down or a bottom up approach to studying design decision making. Third, it makes recommendations for decision support tools and software metrics that wish to support the inherent way that software designers work. Fourth, it makes a tangible contribution to qualitative inquiry in software engineering.

ACKNOWLEDGEMENTS

Frank Maurer, thank you for guiding without pushing, for offering me numerous academic opportunities, for your flexibility in the name of research and for providing me with endless chances to improve. **Mike Chiasson**, thank you for the discussions. You brought qualitative inquiry to life and I left every conversation having learned something new and motivated to pursue the area further. **Guenther Ruhe** and **Sheelagh Carpendale**, thank you for your time and ideas as I prepared for my candidacy exam, and in the shaping of this thesis. **Ron Wardell**, **Helen Sharp,** thank you for your time and insightful questions.

Thank you to **Grigori Melnik**, for always having tea ready, for *really* knowing academic stuff, and for top-notch hugs. **Harpreet Bajwa**, thank you for opening a fabulous door for me. To the **EBE group past and present**, and to the slackers on the other side of ICT 524, thank you for the laughter, the cultural exchange and the research noise.

To the **53 unnamed participants** in this research and the **25 others working with the participants**, thank you so very much for enduring my questions and lurking with professionalism, patience and openness. *This research simply would not exist without you.*

…

Margaret Zannier, thank you for remote cooking, baking and canning lessons, for being a girlfriend and a mother, and for always encouraging me to pursue *my* life. **Danilo Zannier**, thank you for your wit and the laughter it provides, for your ability to focus on the task regardless of surrounding nonsense, and for discussing every dream I ever had with equal excitement and equal responsibility. **Lia Zannier**, thank you for telling me I'm wrong when I need to hear it, for telling me I'm right when I've forgotten it and for always listening, listening and listening some more. To **Patrick Gleeson, Michael Gleeson, Greg, Ladine, Vanessa and Marissa Borschke**, and to **Paula Perry:** thank you for your open hearts and homes in early 2006. To **Ambrosina Zannier** and the **Zannier family,** and to **Helen Borschke** and the **Borschke family,** thank you for all the love and support.

To Mom and Dad

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER ONE: INTRODUCTION

The design of software involves numerous cognitive skills such as mental modeling, mental simulation, problem structuring and decision making, the last of which is the focus of this research. Whether the decision is what to name a method, how to change a class hierarchy or how to organize the underlying architecture of a software application, decision making is prevalent in software design. Whether the decision is which services to expose in an interface, where to place business and presentation logic, or which programming language to use, decision making is dependent upon previous decision making. Whether the decision is how to fix a bug, a debate about design principles or a reconciliation of architecture trade-offs, decision making varies in scope and in the number of decision makers involved. Given the prevalence of decision making in software design, the impact one design decision can have on another, and the number of software designers that can be involved in decision making, it is surprising that the existing literature provides little description about how software design decisions are made. This research addresses this void.

This empirical investigation uses qualitative inquiry, qualitative description and qualitative evaluation to produce an understanding of software design decision making, as it occurs in small software development teams. The inquiry employed semi-structured interviews, on-looker observations and participatory observations. The description employed content analysis, explanation building and the development of case study summaries. The evaluation employed comparisons of decision making theories and action research in a software team. Two main conclusions, a set of smaller results and five descriptive decision models comprise the understanding of software design decision making. This is an understanding of decisions made in software teams of 3 to 20 designers, who use at least some practices and principles defined by agile methods. Fifty-nine case studies, contained in three multi-case studies, constitute this research.

1.1 Purpose of the Research

The purpose of this empirical investigation is three-fold. First, the purpose is to describe: to examine a phenomenon as it occurs in its natural environment, to recognize facets of the

phenomenon as they emerge from the environment, and to develop *an* understanding of the phenomenon for the sake of understanding. The second purpose is to direct improvements in software design decision making: to make salient the inherent tendencies of software design decision making, so that these tendencies can be more easily compared to existing descriptions of decision making and especially to recommendations for software design decision making. The third purpose is to highlight challenges in effecting change in design decision making: to present an experience with action research in software design decision making, as an effort towards improving and changing design decision making.

1.2 Motivation for Studying the Topic

An empirical investigation of software design decision making merits examination for various reasons. There is little descriptive work in design decision making relative to other research topics in software engineering, and more importantly, relative to strong calls to examine this topic [Adelson85, Curtis88, Highsmith02, Highsmith04, Rugaber90, Walz93]. To the best of my knowledge an empirical investigation of software design decision making of the scope of this research has not yet been conducted. In addition, design decision making is ubiquitous in software design [Highsmith02, Highsmith04, Rugaber90]. The decisions made during software design can impact the life of a software product significantly [Highsmith04] yet the software engineering community still understands little about the topic. A third motivator for examining decision making is that decision making is seen, in part, as a social process [Hutchins95, Highsmith04, Kaner96], and the social side of software engineering has gained and continues to gain momentum, especially in the software engineering empirical community [Dittrich07, Demarco99]. Finally, there is significant existing research that presupposes some ideas about decision making and researches decision making using approaches that are quite different than the approach presented in this thesis. For example, the development of decision support tools is a wide area of research that presupposes a specific manner in which software designers make design decisions and develops tools to be applied to a decision environment. Empirically examining the manner in which software designers make design decisions – that is, describing the environment in which decision making occurs prior to tool introduction -

provides a useful comparison to existing but different research approaches to decision making in software engineering.

1.3 Approach to Research

The realized strategy [Mintzberg92] (i.e. what actually occurred) of this empirical investigation was to perform three successive multi-case studies, where I, as the primary researcher, had increasing involvement in the decision making that was examined. In the first multi-case study I interviewed 25 software designers on design decisions they had made. In the second multi-case study I observed decision making at 3 different software organizations, examining 12 decision cases at the first organization, 11 decision cases at the second organization and 5 decision cases at the third organization. In the third multi-case study, I participated in design decision making in a software team over a 6-week period, resulting in 6 decision cases and observation points from attempting to effect change in design decision making. While the specific research technique varied in each of the three studies (i.e. interviews, on-looker observations, participatory observations), I use the term "case study" in each study for five reasons [Easterbrook06] which will be shown throughout this thesis. First, there was a research question established at the beginning of each study. Second, the data was collected in a planned and consistent manner. Third, inferences were made from the data to answer the research question. Fourth, the studies produce a description of a phenomenon. Fifth, threats to validity were addressed in a systematic way [Easterbrook06].

1.4 Outcome of the Research

The outcome of this empirical investigation consist of two main conclusions, a set of eight smaller results, and five design decision models. The two main conclusions are shown across all three multi-case studies. The eight smaller results are distinct from the two main conclusions in that they are shown in one to two of the multi-case studies and they were specific ideas I carried with me from multi-case study to multi-case study. They are "smaller" in the amount of supporting data, but not smaller in value to this thesis. *All the of the results emerged from the data, in no way were they ideas that were generated before the study.* The first main conclusion is that software designers appropriate a solution to a

design decision approximately as frequently as they strive for an optimal or boundedly optimal design solution. That is, they apply a design solution that “just fits” or is “suitable”. The second main conclusion is that the approach a software designer takes in decision making is affected by the software designer’s understanding of the design problem. More specifically, the stronger a software designer’s mental model of a design is, the more apt s/he is to use serial evaluation (i.e. not consider alternatives to a decision problem). The more open a software designer’s mental model of a design is, the more apt s/he is to use consequential choice (i.e. consider alternatives to a decision problem). The eight results found throughout the multi-case studies are below.

1. Time pressure imposes the use of serial evaluation on making design decisions.
2. External goals imposes the use of serial evaluation on design decision making.
3. There is disagreement in the software engineering community about accessing knowledge and using knowledge.
4. Rational decision making, naturalistic decision making and real options theory were inaccurate descriptions of software design decision making for the decision making cases that comprise my data.
5. Explanation based decision making is a good “start” for describing design decision making.
6. Agile environments lead to the use of consequential choice more than non-agile environments.
7. When effecting change in decision making, if a group of people meet in a room they should a) all understand why they are there and b) see tangible value in their participation.
8. Change management requires explicit, unwavering support at the senior levels of the organizational hierarchy.

Finally, the five decision models are as follows:

1. Appropriating, Single Designer, Small Problem Decision Making
2. Appropriating, Two to Five Designers, Small Problem Decision Making
3. Reasoning, Single Designer, Medium Problem Decision Making
4. Reasoning, Two to Five Designers, Medium Problem Decision Making
5. Explaining, Five to Ten Designers, Large Problem Decision Making

Each of these models and their respective components (e.g. Small Problem Decision Making) will be defined in the following chapters.

1.5 Initial Definitions

The word “design” is used throughout this thesis, in the context of software design. Thus, a definition is imperative. From [Ghezzi91], software design is defined as,

> A system decomposition into modules – description of what each module is intended to do and of the relationship among the modules. Such a description is often called the software architecture, or the software structure. … We can view design as a process in which the architecture is described through steps of increasing detail. [Ghezzi91, p.64].

From [Schach99], design is separated into architectural design and detailed design. Architectural design is, “(also known as general design, logical design or high-level design) a modular decomposition of the product… the specifications are carefully analyzed, and a module structure that has the desired functionality is produced.” [Schach99, p 404]. Detailed design is, “also known as modular design, physical design or low-level design, during which each module is designed in detail.” [Schach99, p. 404]. From [Poppend03], “design is a problem solving process that involves discovering solutions through short, repeated cycles of investigation, experimentation, and checking the results.” [Poppend03, p. 19]. This last definition differs from the first two in its recognition of ill-structured problems and an “unstructured” approach to producing software design where “designers cycle between high-level design and detailed solutions to produce a design”. [Poppend03, p 19]. Given these definitions, this thesis recognizes that software design occurs at varying levels of abstraction, but the boundaries between these varying levels of abstraction are blurry and context dependent.

In addition, this thesis continuously compares rational approaches to decision making, to naturalistic approaches to decision theory, and thus definitions of each are required. I define rational approaches to decision making as descriptions and recommendations for decision making that are rooted in detailed examinations of decision alternatives,

mathematical calculations to select an optimal decision alternative among many decision alternatives, and the idea that the decision maker has adequate time to compare and contrast decision alternatives [Simon55, Orasanu93]. I define naturalistic approaches to decision making as descriptions and recommendations for decision making that recognize situational behaviour, real-time scenarios potentially impacted by time pressure, and the expertise that a decision maker brings to decision making [Flyvberg01, Klein93b].

1.6 Chapter Summary

This thesis contains 12 chapters. Chapter 2 is a literature review describing philosophical underpinnings of this research, the idea of problem solving in software engineering, the perspectives on performing software design as a science or as an art, how software design can be measured, how cognition is captured in software design, an overview of decision making and finally, an overview of the state of empiricism in the software engineering community. Chapter 3 introduces the empirical studies performed, their overlapping purposes, assumptions and definitions and their different execution. Chapters 4 and 5 address the first empirical study, which used interviewing to describe design decision making as it relates to rational and natural decision making. Chapter 4 describes the techniques used in this study and my experiences with qualitative inquiry. Chapter 5 provides the results of this first empirical study. Chapters 6 and 7 address the second empirical study, which used on-looker observations to describe design decision making as it relates to real options theory. Chapter 6 describes the techniques used and Chapter 7 provides the results. Similarly, Chapters 8 and 9 address the third empirical study, which used participatory observations to describe design decision making as it relates to explanation based decision making and effecting change in a decision making environment. Chapter 8 introduces the techniques used in this study and Chapter 9 provides the results of this study. Chapter 10 provides a cross case comparison of all three studies and describes the five resulting design decision models. Chapter 11 discusses validity. Chapter 12 concludes this thesis with a look at the impact of this work on the software engineering community.

CHAPTER TWO: LITERATURE REVIEW

Seven topics are pertinent to a review of software design decision making and are presented in increasing order of relevance to this research. The topics are listed in Table 2.1, in the leftmost column. Upon researching these topics a common theme emerged: the ideas found within each topic were shaped by two *seemingly* opposing perspectives. For example, rational decision making is one perspective from which to view decision making; naturalistic decision making is the other perspective. It can be claimed that the definition of naturalistic decision making opposes the definition of rational decision making because of the concepts that exist in each decision making approach (e.g. naturalistic decision making emphasizes satisfactory decisions, rational decision making emphasizes optimal decisions).

It is not the intent of this thesis to prove a perspective as being right or better than another perspective, or to simply dichotomize the topics. In fact, this review shows some blurring between the two perspectives, for each topic. The two perspectives were created to provide differing perspectives that act as a platform from which to discuss the ideas presented in each topic. An exhaustive review of perspectives within each topic is beyond the scope of this research.

Among the topics there is similarity in the ideas inherent in the perspectives. For example, the ideas found in the perspective called positivism are used by the ideas found in the perspective called quantitative. The format of Table 2.1 suggests similarity among the perspectives listed in the same column. Again, it is not the intent of this thesis to prove these perspectives as being similar. Similarity is acknowledged but discussing the details of this similarity is beyond the scope of this research. With these points in mind, the literature review follows.

Table 2.1: Pertinent Literature Review Topics

Topic	Perspective A	Perspective B
Reality and Knowledge	Positivism	Interpretivism
Problem Solving	Well Structured Problems	Ill-Structured Problems
Doing Software Design	Science	Art
Measuring Software Design	Objective	Subjective
Software Design Cognition	Explicit	Implicit
Decision Making	Rational	Natural
Empirical Studies	Quantitative	Qualitative

2.1 Reality and Knowledge

Two theoretical perspectives are the platforms for an epistemological debate about reality and knowledge, and are the underlying perspectives through which this research is examined. These perspectives are called positivism and interpretivism.

Positivism holds that in empirical research a researcher and the object of inquiry are independent entities [Orlikowski91]. Facts and theoretical propositions are separate from the state of individuals involved (in any way) in the research [Patton02, Lee91]. Natural science methods provide an understanding about reality via objective and verifiable knowledge [Orlikowski91, Buchanan98]. Positivism assumes there is a reality to describe and that objective and universal knowledge of that reality can be achieved [Buchanan98]. Alternate terms for positivism are: objective, nomothetic and quantifiable. These terms do not carry the philosophical weight that the word positivism carries, but they are suggestive of concepts upheld by positivism.

Interpretivism holds that in empirical research a researcher and the object under study are incapable of being understood independently [Orlikowski91]. Reality and knowledge of that reality are social products and cannot be understood independent of the individuals that construct that reality. [Orlikowski91, Lee91] Natural science methods are inadequate to study a social reality [Lee91]. The important reality in interpretivism is “what people perceive it to be.” [Patton02, p.69]. Alternate terms for interpretivism are: subjective,

idiographic, and qualifiable. These terms also do not carry the philosophical weight that the word interpretivism carries, but are suggestive of concepts upheld by interpretivism.

The conceptual separation of positivism and interpretivism is not entirely incompatible, at a practical level. "Objectivity of any particular knowledge is subjectively rooted in the belief of a certain community of individuals that the knowledge they share describes reality objectively." [Buchanan98]. In practice, empirical research techniques belonging to each perspective are integrated [Orlikowski91, Lee91, Mingers01, Landry92].

2.2 Problem Solving

The extent to which a problem is structured establishes two perspectives from which to view a problem. These perspectives are a well structured problem and an ill structured problem. They can be viewed at an abstraction level that considers problems in general and at an abstraction level that considers software development problems.

A well structured problem is a problem that has criteria that reveal relationships between the characteristics of a problem domain and the characteristics of a method by which to solve the problem [Simon73]. "There is at least one problem space in which can be represented the initial problem state, the goal state, and all other states that may be reached, or considered, in the course of attempting a solution to a problem" [Simon73, p. 183]. An example of a well structured problem is a game of chess.

An ill-structured problem is a problem that is not well-structured [Simon73]. An ill structured problem requires problem structuring, where a problem solver converts an ill structured problem to a well structured problem [Simon73]. A problem solver spends more time in problem structuring than in actually solving a problem once it is structured [Simon73]. An example of an ill structured problem is the design of a new house [Simon73]. Decisions made in early design sketches of the house establish structures under which following decisions will be made [Simon73].

Specific to software development, a well structured problem provides a platform for Taylorism. Taylorism is a management solution. Factories are managed through scientific methods rather than by the use of "the rule of thumb". Scientific selection of the workperson, task breakdown and separating planning from execution are three characteristics of Taylorism [Taylor07]. Taylorism is a potential foundation for software engineering practices that *oppose* agile practices [Agile07].

Specific to software development, an ill structured problem provides a platform for wicked problems. Wicked problems are the types of problems that planners solve. The formulation of a wicked problem is the actual problem; wicked problems have no stopping rule; solutions to wicked problems are not true or false, but good or bad [Rittel73].

The dichotomy between well structured problems and ill structured problems is not entirely incompatible. "There is no real boundary between well structured problems and ill structured problems." [Simon73]. Given the definitions of the two, much of problem solving can be viewed as problem structuring and vice versa.

2.3 Doing (Software) Design

Design as science versus design as craft outline two perspectives from which to view software design. They can be viewed at an abstraction level that considers design in general and at an abstraction level that considers *software* design.

Design as Science views design as a partially scientific discipline which separates design from the product designed [Horvath04, Lea93]. Design as Science uses analytic models and assumes that the problem to be solved (i.e. designed) is described in a comprehensive and precise manner [Horvath04, Lea93, Lowgren95]. Design as Science suggests a step-by-step approach to design [Royce90].

Design as Craft views design as a merger of a design setting and a design product [Lea93, Lowgren95]. Design as Craft uses the ability of humans to achieve a desired design, [Wardell91], because contemporary design often fails to improve the human condition

[Lea93]. Design as Craft acknowledges the limitations of step-by-step approaches to design [Royce90, Wardell91].

Specific to software development there is a debate between software "engineering" as engineering or as an art. The waterfall process receives much criticism because it advocates an engineering approach to design [Royce90]. The pursuit of requirements as a formal specification is also popular [Parnas86]. Outright claims that software development should follow engineering principles can also be found [McConnell98]. The other end of the spectrum acknowledges software development practices as insufficient for software design [Lowgren95]. In fact, the publication of the waterfall method acknowledges the limitations of such a step-by-step process [Royce90].

The dichotomy between Design as Science and Design as Craft merges via somewhat fuzzy boundaries of engineering design [Horvath04, Royce90]. "In panic people try to replace the lost order of the organic process, by artificial forms of order based on control." [Alexander79].

2.4 Measuring Software Design

When measuring software design the literature shows two perspectives from which to begin. They are quantifiable metrics and qualifiable metrics.

Quantifiable software design metrics must accurately represent the attributes they are supposed to quantify [Kitchenham02]. Consequently, much work teaches the proper use of metrics [Fenton97]. It is even stated that software engineering can not become a true engineering discipline, as mentioned in Section 2.2, until the field establishes proper measurement theories [Fenton97]. In practice, consistent relationships between internal and external software design attributes are exceedingly difficult to develop [Briand96]. This is because software does not directly show design attributes but only exhibits characteristics indicative of design attributes [Briand96]. This is also because "good" software design is not consistently defined [Briand01]. Dating back at least 25 years,

researchers have tried to develop a consistent and agreed upon set of design metrics, to no avail.

Qualifiable software design metrics relies on more elusive concepts such as quality or goodness, and rely on human judgment of "good" [Dromey95]. Recognizing that "good" software design is undefined, [Fowler97], qualifiable software design uses the concept of "good enough" software, to evaluate a software design. Additionally, qualifiable software design metrics rely on design patterns as the cornerstone of good software design [Gamma95]. Software design patterns provide common solutions to common design problems but still require a designer to tailor a design pattern to a specific design solution [Gamma95].

Reconciling quantifiable and qualifiable software design metrics is difficult because it is akin to matching internal product characteristics to external design attributes. As of yet, this has not been accomplished.

2.5 Capturing (Design) Cognition

Capturing software design cognition (i.e. the way a software designer thinks) can be done explicitly or implicitly and much research exists in both areas. The use of design rationale approaches is an explicit approach to capturing design cognition. Arguably, design rationale approaches capture the rationalization of a design, more than the way a software designer thinks. Empirically studying software designers at work is an implicit approach to capturing design cognition.

2.5.1 Explicit Capture: Design Rationale

Design rationale is the documenting of design decisions and design decision justifications [Lee96, Klein93]. The purpose is to facilitate reasoning and communicating about a design as well as organizational learning [Lee96, Klein93]. Two approaches are examples of the attempts to solve the issue of capturing design decisions without intruding too heavily on the design process. The first approach stems from an Issue-Based Information System (IBIS) defined by Rittel (1970). The IBIS model partitions wicked problems into Issues,

Positions and Arguments. An issue is a point or concern regarding a design problem, and contains one or more positions that make a statement about an issue. Positions contain one or more supporting or refuting arguments [Conklin88]. A graphical tool, gIBIS, has also been developed, implementing IBIS to understand the structure between and within design decisions [Conklin88]. Procedural Hierarchy of Issues (PHI) also extends IBIS by including issues that are not deliberated and by introducing a serving relationship between issues. "Issue A serves Issue B if the answers to A influence what answers are given to B" [Fischer89]. A Design Representation Language based on IBIS [Lee96] proposed that a design problem consists of alternatives, goals, claims, questions, group, viewpoint, procedure and status. The second approach to modeling design rationale amalgamates Questions, Options and Criteria (QOC) [MacLean96]. A decision alternative exists in a space of possibilities and design decisions are justified through questions, options and criteria.

External evaluations of design rationale have not been positive. An initial investigation attempted to determine the usefulness of design rationale documents [Karsenty96]. The conclusion of the study was that design rationale documents should be useful to interested designers; however, design rationale documents were, at the time, not sufficient, as less than half of the designers' design rationale questions were answered by the design rationale documentation. A second study investigated the usability of QOC [Shum96]. This study critiqued QOC as lacking vocabulary, providing a poor representation of dependencies between decision problems and being too restrictive [Shum96]. Lastly, speaking to the motivation behind design rationale, one investigation found software design meetings to be structured suggesting design meetings did not require structure or organization. This is challenges the usefulness of design rationale models as a method to organize design [Olson96].

2.5.2 Implicit Capture: Design Studies

A survey of design studies finds that the following six related qualities impact software design: expertise, mental modeling, mental simulation, continual restructuring, preferred

evaluation criteria and group interactions. Each quality is defined below, and the results of the studies pertaining to each quality are summarized.

Expertise is the knowledge and experience software designers have in design [Adelson85]. The issue of expertise, or knowledge, is fundamental to software productivity and quality [Curtis88]. "Although individual staff members understood different components of the application, the deep integration of various knowledge domains required to integrate the design of a large, complex system was a scarcer attribute" [Curtis88]. Individual team members to not possess enough knowledge for a project and must learn additional information before they are productive [Walz83].

A study of three expert designers and two novices required all participants to design an application to the best of their abilities, and varied two qualities of the task given to the participants: the domain of the object to be designed and the object itself. Three scenarios emerged: Familiar Domain & Unfamiliar Object, Unfamiliar Domain & Unfamiliar Object and Familiar Domain & Familiar Object [Adelson85]. The last combination, Unfamiliar Domain & Familiar Object, does not exist [Adeslon85]. “The domain of the object corresponds to the general classification of the system according to its use.” The object is a specific entity in the domain. [Adelson85]. In the first scenario, experts formed an internal model and ran mental simulations. The models progressed from abstract to concrete. In the second scenario, experts created a mental model but did not focus on implementation details and made a mistake in moving from abstract to concrete models. In the third scenario experts formed a mental model early, made detailed drawings and ran very few mental simulations. In all three scenarios, the novices considered one task at a time, did not have a simulate-able model, and had low-level representations of programming knowledge. The expert designers had a minimum of eight years design experience and were considered expert in the design of communication systems. The novices had several years of programming experience but less than two years as designers.

A second study examined three expert designers and evaluated their approach to design as well. The study found that top-down design could be achieved only after some bottom-up

thinking, trial design, coding and backtracking had occurred [Guindon90b]. The subjects of the study, all considered expert, had the following backgrounds: The first designer had a software engineering master degree, was top student of his class and had five years professional experience. The second designer had a doctoral degree in electrical engineering and ten years professional experience. The third designer temporarily suspended doctoral work in computer science to work in industry for three years.

A third study examined the differences between novice designers and expert designers outside of software engineering [Ahmed03]. Novice designers collected drawings and documentation more frequently than experienced designers; experienced designers combined activities more often and with more activities than novice designers. Six experts participated in the study, with 8 to 32 years experience and 3 days to 56 weeks on the examined project. Six novices participated in the study, with 3 months to 2 years experience and less than 8 weeks on the project.

Mental modeling is the internal or external model a designer creates that is capable of supporting mental simulation [Adelson85]. "A model is an abstract representation of a system that enables us to answer questions about the system" [Bruegge04, 12]. External modeling is the development of diagrams, code or any physical model of the application [Ahmed03]. Internal modeling is the mental development of a model of the application [Guindon90a].

Expert designers incorporate the recognition of solutions varying in abstraction into their design problem solving [Guindon90a]. In one study, experts working in a familiar domain with an unfamiliar object formed mental models for their problem [Adelson85]. In an unfamiliar domain with an unfamiliar object, the experts created mental models lacking in detail, constraints and labeling. In the design of a familiar domain and familiar object, a detailed mental model developed early in the design process [Adelson85, Curtis88]. Novices, on the other hand, represented plans as pseudocode, which is considered to be, at best, a loose model [Adelson85, Sonnetag98].

Individuals maintain a mental model that is used in the creation of a group model [Gasson98]. The development of a group model is complete when most individual models sufficiently match the group model [Gasson98]. A model develops as team members learn from one another about the application and required computational structures [Walz93].

Mental simulation is the "ability to imagine people and objects consciously and to transform those people and objects through several transitions, finally picturing them in a different way than at the start" [Klein98]. External and internal modeling provide a foundation upon which mental simulations run. Designers alternate between concrete scenarios and abstracting from these scenarios [Gasson98].

In the study by Adelson and Soloway (1985), the internal model of a familiar domain and unfamiliar object formed a basis for mental simulations by experts, and in an unfamiliar domain with an unfamiliar object, mental simulations were still attempted, but were attempted in isolation and incorporated fewer constraints [Adelson85, Curtis88]. In familiar domains with familiar objects, mental simulations did not occur as frequently with expert designers in comparison to unfamiliar domains or unfamiliar objects [Adelson85]. Novice designers, however, did not always maintain a model sufficient for mental simulations [Adelson85].

In a second study, "the simulations of problem domain scenarios are not found in a monolithic block preceding design as would be prescribed by a waterfall model. Rather, they are found interleaved with solution development throughout the design session" [Guindon90a]. When mental simulations based on internal models are cognitively taxing, external models formed the basis for the mental simulation or the mental simulations were shallow [Guindon90a]. The designer's mental model of the solution, external representations and simulations constituted part of the designer's knowledge [Guindon90a]. Consideration of alternative solutions in mental simulations rarely occurred and quick selection of alternatives marked these situations [Guindon90a].

Further evidence showed that limitations of working memory may hinder mental simulation capabilities [Kim92]. While the skill of simulation was seen as independent from the skill of creating a model, [Adelson85], the quality of the model dictates the ability to execute a simulation on that model [Adelson85].

Continual restructuring is the process of turning an ISP to a WSP. Continual restructuring is also known as systematic expansion [Adelson85, Guindon90b]. "The background conditions in the social world are not physical facts. Nor are they psychological facts about what individuals desire and what they believe to be rational. The background conditions are patterns of behaviour which are characterized by expert exercise of tacit skill" [Flyvberg01, 45]. The concept of continual problem restructuring embraces this quotation, emphasizing the importance of the context of any situation in evaluating its execution.

In one study, experienced designers exploited the context of the current state of design to determine relevant issues [Ahmed03]. In another study, design element properties and design organization impacted the context of the design [Malhotra80]. Even in an algorithm design study, discovery involved a prepared problem and acknowledgement of the task-domain space [Kant84]. Evidence suggested that the context of the design consisted, at least in part, of organizational mechanisms and structures that designers exploited to frame design problems. The designers revisited the framed problem as new context-based information emerged [Gasson98, Walz93]. This process has been termed "organized anarchy" [Clegg94].

Experts in familiar domains with unfamiliar objects, or unfamiliar domains with unfamiliar objects continually restructured problems, with decreasing levels of success according to decreasing expertise [Adelson85]. The structure of a design has been shown to exist on a continuum, and a designer's knowledge of a problem imposes structure onto that problem [Guindon90b].

The term "*preferred evaluation criteria*" refers to the minimal criteria a subject adopts to structure an ISP and guide the search for a satisfactory solution [Guindon90a].

A subset of problem restructuring is preferred evaluation criteria [Guindon90a]. In one study, the preferred evaluation criteria occurred on an individual level, in that loosely defined objectives guided action [Gasson98]. In another study, the preferred evaluation criteria occurred on a group level, in that team members meeting to discuss design arrived with individual ideas, but ultimately a few members convinced the group to focus on a small set of objectives [Walz93, Curtis88].

Caution must be taken in selection of preferred evaluation criteria. Properly chosen criteria reduced design problem complexity, but poorly chosen criteria led to early reduction of the design problem and a closed minded approach to the situation [Guindon90a, Gasson98, Walz93]. The preferred evaluation criteria generated concern similar to that of confirmation bias, tunnel vision and groupthink [Vincente03, Norman93].

Group interactions is the dynamics of group work in software design. Motivation for the examination of group interactions arises from vast amounts of time designers spend in informal communication [Herbsleb03]. The terms "distributed" and "shared" cognition suggest that individual mental models coalesce via coordinated group action, resulting in a common model [Gasson98]. The coalescing may result from the domination of a small coalition of individual designers (or sometimes just one designer) controlling the direction of the project [Curtis88]. In relation to the preferred evaluation criteria, one study showed this control can result in success or failure of a closed-minded group upon the arrival of new design ideas [Walz93].

Reconciling explicit design cognition capture and implicit design cognition capture is not well researched in software development, perhaps because it is akin to evaluating how people work versus how they say they work: a topic for the social sciences. Such research is beyond the scope of this work.

2.6 Decision Making

Two approaches to decision making provide the perspectives from which to view multiple decision making theories as well as the perspectives from which to view software design decision making. The two decision making approaches differ in their emphasis on mathematical foundations and real-time scenarios. They are called rational decision making and naturalistic decision making.

2.6.1 Rational Decision Making

A rational decision "is one that conforms either to a set of general principles that govern preferences or to a set of rules that govern behaviour. These principles or rules are then applied in a logical way to the situation of concern resulting in actions which generate consequences that are deemed to be acceptable to the decision maker" [Stirling03, 5].

Rational decision making is characterized by an appreciation for mathematical computation and comparison of decision alternatives. A rational decision is also known as a normative decision and is prescriptive. A normative decision "prescribes for given assumptions, courses of action for the attainment of outcomes having certain formal 'optimum' properties" [Luce58, 63]. "Prescriptive" is defined as recommending an approach to decision making to the decision maker.

A rational decision has three features that are parts of the decision maker's approach to decision making [Simon55, Siddall72].

- **Decision Alternatives**. The decision alternatives are represented by a set of possible courses of action and potential outcomes for each action.
- **Utility Function**. A utility function assigns a value to each possible action based on its outcome.
- **Probabilities**. A decision has information or probabilities as to which outcome will occur in case of the selection of a particular alternative.

Rational decision making is limited by three assumptions. The first is that a set of possible courses of action and the probability of outcomes is known. The second is that the decision

maker's goal is to optimize. The last assumption is that combinatorial explosion of alternatives and the time involved in mathematical calculations of situations of combinatorial explosion are not a large concern [Orasanu93, Klein98].

Multi-Attribute decision theory (MAUT) is a generic rational decision theory process in which a decision maker specifies all possible outcomes and ranks them according to objective criteria, external to the decision maker's value system [Doherty93]. A prime example of MAUT is the Analytical Hierarch Process (AHP). AHP is a popular multi-attribute rational decision theory approach developed by Thomas Saaty [Saaty80]. A decision maker specifies the possible outcomes of a decision and partitions the decision problem into attributes representing the decision problem. In software design, an example would be evaluating performance metrics on various search algorithms and picking the search algorithm that produces the maximum AHP outcome.

Real Options Analysis (ROA) is a financial model of decision making [Lasher2006]. ROA's fundamental purpose is to provide an approach to considering alternatives in a decision, by waiting to make a decision [Lasher2006]. ROA is rooted in rational decision theory in that it specifies a decision alternative and places a cost value on that alternative. In ROA, "a real option is a course of action that can be made available, usually at a cost, which improves financial results under certain conditions" [Lasher2006]. An option is viewed as an asset, and investing in that option is viewed as investing in flexibility [Erdogmus2003, Lasher2006]. An example is as follows: A buyer debates whether or not to purchase a piece of property for some price *x*. Instead of forcing the decision to buy immediately, the seller offers the buyer the option of purchasing the property at cost *x* until some maturation date. For this option, the buyer pays a price immediately to the seller. If, between the purchase of the option and the maturation date, the market value of the property rises above cost *x*, the buyer earns money by purchasing the property at the lower option price (if s/he is able and interested). If, between the purchase of the option and the maturation date, the market value of the property drops below cost *x*, the buyer can decide not to exercise the option to buy the property at the option price, because s/he can get it at a cheaper price. In the end, the buyer only loses the cost of the option (and its interest).

To determine the cost of the option, five parameters and a valuation method are required. The five parameters are as follows [Lasher2006].

a. The value of the underlying risky asset (e.g. the property value)

b. The exercise price (e.g. *x*)

c. The time to expiration of the option (e.g. maturation date)

d. The standard deviation of the underlying risky asset (e.g. how much the property value will change)

e. The risk free rate of interest over the life of the option (e.g. money not invested in the property until the maturation date can be earning interest).

Historically, static net present value (NPV) was used to estimate the value of an option but it is now widely accepted that NPV underestimates the value of flexible assets [Sullivan 1999]. In software design, an option could be a decision to invest in a particular platform functionality so that other functionality could be constructed in the future, at a cheaper and certain price. If later on the company decides they don't want this functionality, this functionality can be abandoned. Another example would be to decide to use a given programming language, that is new (e.g. when XML was first emerging). A software company might invest some research and resources into the language but wait until the language's second version is available to decide whether or not to commit to the programming language.

2.6.2 Naturalistic Decision Making

A naturalistic decision "connotes situational behaviour without the conscious analytical division of situations into parts and evaluation according to context-independent rules" [Flyvberg01, 21]. Naturalistic decision making is characterized by an appreciation for real-time scenarios and judgment biases. A naturalistic decision is defined by the following six characteristics [Klein93b]:

- Manifests itself in dynamic and continually changing conditions.
- Embodies real-time reactions to these changes.
- Embraces ill-defined tasks and goals.
- Resolves itself under the guidance of knowledgeable decision makers.

- Utilizes situation assessment over consequential choice.
- Has a goal of satisficing instead of optimizing.

Three terms in these six characteristics require definition: consequential choice, serial evaluation and satisficing.

- **Consequential Choice**. The organized analysis of options and potential outcomes, typical of rational decision theory [Lipshitz93].
- **Serial Evaluation**. The absence of choice in real-world decisions. A decision maker exercises an action that may or may not be considered amongst a set of choices [Lipshitz93]. A decision maker thinks of a possible option then uses mental simulation to evaluate the option. If the mental simulation succeeds the decision maker executes that option.
- **Satisficing**. The acceptance of a satisfactory situation [Klein98].

A key point of serial evaluation is that it aligns with satisficing [Klein98]. One design solution is considered at a time and once a satisfactory solution has been found, evaluation of alternatives stops, regardless of the quality of that satisfactory solution [Klein98].Phrased another way, serial evaluation involves no evaluation of trade-offs, as occurs in consequential choice [Luce58, Klein98].

Naturalistic decision theories are limited by the absence of a definable, predictable outcome. There are 10 approaches to naturalistic decision making. The decision making approach and a brief explanation of each are below.

- **Requisite Decision Model**. A shared social reality represents a decision problem and is requisite if the participants of the shared social reality deem the form and content of the model to be sufficient to solve a problem [Phillips84]. In software design an example of a requisite reality is when a group's shared social reality is rich enough to produce adequate software systems that are able to address the complexity of the functionality required for a customer.
- **Prospect Theory**. Prospect theory introduces the concept of the certainty effect, which states that a decision maker will select more certain situations when possible

gain is large, even if the certain situation results in a smaller gain than the alternative. When the situation is not as certain the decision maker will opt for higher gain [Kahnema79]. An example of using Prospect Theory in software design is copying and pasting code from one section of the system to another, because the developer knows the code works and does not have time to consider refactoring the common code.

- **Signal Detection Theory**. In Signal Detection Theory information from the external world and a decision maker's criterion for judgment result in one of four possible situations: hit (decision maker correctly identifies signal present), miss (decision maker incorrectly identifies signal absent), false alarm (decision maker incorrectly identifies signal present), correct rejection (decision maker correctly identifies signal absent) [Cooksey95]. In software design an example of Signal Detection Theory is in debugging an application, when the developer thinks s/he has the explanation for the existing bug, changes the code accordingly, and re-tests the system to confirm is s/he was right. Another example is when the decision maker builds functionality that is required (hit), does not build functionality that is required (miss), builds functionality that isn't required (false alarm), and doesn't build functionality that isn't required (correct rejection).
- **Elimination by Aspects**. Elimination by Aspects examines uncertainty through probabilistic elimination of alternatives via subjective aspects. Each alternative has a set of aspects, repeatedly chosen with a probability that is proportional to its weight. [Tversky72]. In software design an example of Elimination by Aspects is when a developer tries to narrow down the location of a bug by asking a user what they were doing when the bug occurred. Another example is the elimination of certain development platforms because they will be unable to meet the basic performance requirements for the software.
- **Social Judgment Theory**. Social Judgment Theory defines a Cognitive Continuum Index (CCI) which fixates on the conflict between intuitive and analytical cognition. Intuition and analysis bound either end of a continuum, converging on quasi-rationality [Brehmer88]. An example of these problems would be bug fixes found by stepping through code with a debugger and watching the value of a parameter

change. In contrast, some design problems would fall on the end of the cognitive continuum that is labeled “Intuition”. An example of these problems would be how to best model the customer’s domain. An example of quasi-rationality is identifying a recurring problem in code, finding a new technology to implement that will resolve the problem and selling the use of the new technology to business managers.

- **Heuristics and Biases**. Heuristics and Biases is an examination of the heuristic principles used to reduce complex tasks in uncertainty. Three factors affect decision making: representativeness, availability, and anchoring and adjustment [Tversky82]. An example of the use of availability is when a developer searches on the internet for a piece of code to solve his/her problem and takes the first piece of code listed in the search. An example of the use of anchoring and adjustment is when a developer starts with an existing design (either from past experience, internet searches, discussions with someone who has written similar code, or the like) and modifies the code to suit his/her situation.
- **Dominance Structuring**. Dominance Structuring utilizes bolstering and deemphasizing tactics to select an alternative subjectively dominant above all others. Bolstering is the emphasis of positive traits of an object. Deemphasizing is downplaying of negative traits of an object [Montgom93]. In software design an example of the use of bolstering is emphasizing how a class will improve the cohesion of an existing class. An example of the use of deemphasizing is to downplay the coupling that this may introduce between two classes, depending on the situation.
- **Explanation Based Decision Making**. Explanation Based Decision Making aligns a story of an event with an alternative. The primary use of explanation based decision making is in jurors [Penning93]. In software design an example of the use of Explanation Based Decision Making is a design outcome which can best be explained as suitable using a particular story. The diagram for Explanation Based Decision Making is found in Figure 2.1. A decision maker constructs an explanation (or story) based on evidence, world knowledge about similar events, and knowledge about explanation structures in the decision domain. For example, the decision to

choose agile programming over other programming approaches may be because it is considered the most flexible within a fast-changing industrial environment.

- **Participatory Decision Making**. Participatory Decision Making defines divergence (team members are separated on an issue), groaning (team members vocalize unhappiness with an issue) and convergence (team members come together on an issue) as three required zones of group decision making which values full participation, mutual understanding, inclusive solutions and shared responsibility among decision participants [Kaner96]. In software design an example of the use of Participatory Decision Making is a design meeting where independent architectural decisions are attempting to reconcile, and team members have opposing viewpoints for alternatives in dependent decisions. The team identifies their differences (divergence), argues over them (groaning) and ultimately resolves them (convergence).
- **Recognition Primed Decision Making**. Recognition Primed Decision Making (RPD) epitomizes naturalistic decision making via situation assessment, serial evaluation, changing environment and ill defined goals and tasks [Klein98]. In software design an example of the use of Recognition Primed Decision Making is when a developer thinks of a possible solution to an aspect of a design problem, implements it, tests it, and when it works, moves onto the next task.

Reconciliation between rational decision making and naturalistic decision making occurs throughout the definitions of decision making methods. For example, in MAUT, the decision maker assigns subjective weights to attributes. In Elimination by Aspects, aspects are given weights to aid in selection.

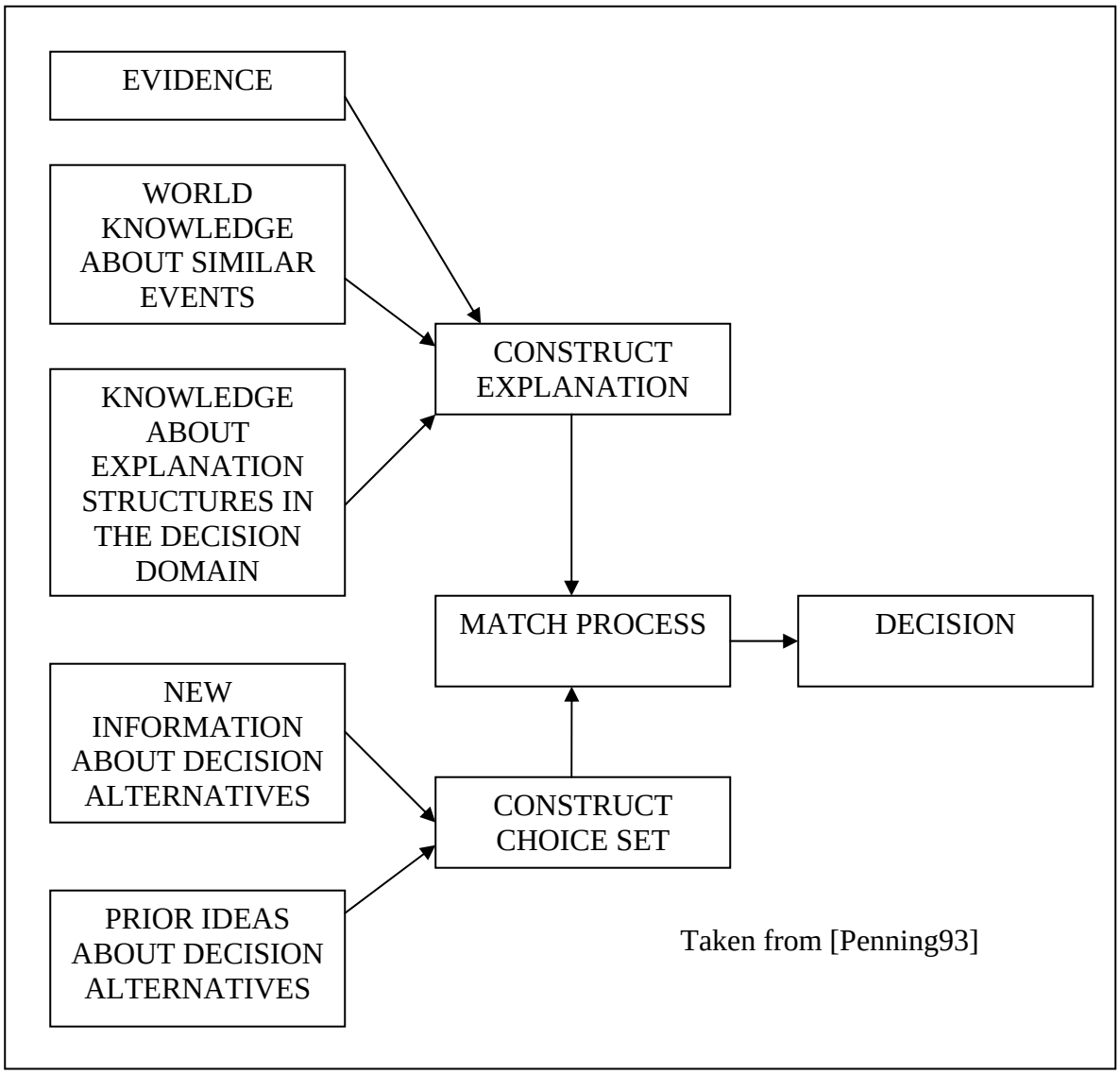

Figure 2.1 Diagram of Explanation Based Decision Making

2.7 Empirical Studies

The importance of empirical studies to software development has been made clear, [Basili86, Basili92, Basili96, Basili99, Perry00, Juristo01, Kitchenham02, Tichy98, Walker03], and within this area of study there are two perspectives that are rooted heavily in positivism and interpretivism [Dawson03]. They are quantitative empirical studies and qualitative empirical studies. Similar to positivism and interpretivism, they also can be examined from the perspective of natural science empirical methods and social science empirical methods, respectively.

Much of software engineering experimentation is quantitative [Juristo01]. Quantitative empirical research uses laboratory based experiments, quantitative metrics, hypothesis testing and simulations to build a body of knowledge around a specific area of study [Juristo01, Briand99b]. The use of quantitative empirical studies is well developed as a result of the strength of natural science research in general.

Qualitative empirical studies take place in the context of a real life situation, leveraging context as critical to understanding [Patton02]. Qualitative empirical studies are typically focused on people as the object of study. That is, the object of study is a subject [Flyvberg01]. Subjects have the opportunity to thoroughly and accurately detail their points of view concerning the topic at hand [Patton02]. Results from qualitative empirical studies are derived from three types of data-collection: interviews, observation and documents. A potential weakness of qualitative experimentation is the impact of the researcher on the experimentation and thus, on the results [Flyvberg01, Patton02]. Qualitative designs are naturalistic: They occur in real-life settings without manipulation of the event being studied. As a result of this naturalistic quality, the entire design cannot be given in advance; it must be allowed to emerge [Patton02].

Merging quantitative empirical studies and qualitative empirical studies is gaining acceptance. Much work in information systems supports a pluralism of the two practices with an emphasis on whichever is more appropriate in a given context [Orlikowski91, Lee91, Mingers01, Landry92]. In software development the number of publications advocating qualitative studies and the number of publications using qualitative studies are increasing. A review of both subtopics is below.

The use of qualitative studies in software development borrows methodologies from the social sciences, necessitating work that educates the software development field on such methodologies [McGrath95]. Such education began by first recognizing the impact that the human factor has on software development [Basili96, Wohlin04]. Workshops such as [Kemerer97] compare quantitative and qualitative studies to identify the suitability of each

perspective and their potential integration. The integration of the two perspectives, as a result of the mix of technical and human factors in software development is advocated again in [Seaman99]. The ethics of qualitative studies, which often involve field work in software organizations that require confidentiality, is addressed in [Goterbarn01]. The use of case studies, a popular approach to qualitative studies (as will be apparent in the following paragraph), motivates publications that teach the details of case studies [Yin02, Bratthall02, Perry04]. Specific techniques within qualitative studies for data collection (e.g. interviews, observations [Patton]) and data analysis (e.g. content analysis [Kripend80]) are also available [Seaman99, Lethbridge05, Karahasanovic05]. The last decade shows much promise for the use of qualitative studies in software development.

The use of case studies as a qualitative approach is extremely popular. For example, case studies have been used to evaluate programming productivity, software maintenance organizations, distributed collaboration, impact analysis, requirements change processes and specifications, technical review meetings, software process improvement, knowledge management and cost estimates [Jeffery79, Seaman96, Bellotti96, Emam97, Robillard97, Laitenberger02, Dingsoyr02, Beecham03, McDonald05]. This shows the diversity of case studies as an approach to qualitative studies. Examples of the use of qualitative data in empirical studies outside of case studies are [Iivari96, Briand99 Cockburn01, Herbsleb01]. As many of these qualitative studies are published in highly respected journals such as Empirical Software Engineering, the importance of qualitative studies is clear and the integration with quantitative studies is well underway.

2.8 Chapter Summary

The seven topics, Reality and Knowledge, Problem Solving, Doing Software Design, Measuring Software Design, Software Design Cognition, Decision Making, and Empirical Studies can be viewed as having extreme and opposing perspectives. Across the topics there is an overarching debate between natural science research and social science research. It is important that the differing perspectives be made explicit so that this research on software design decision making, which continuously integrates differing perspectives, can be more easily understood.

CHAPTER THREE: INTRODUCING THE EMPIRICAL STUDIES

Three studies similar in purpose and scope, but different in execution, are the anchors of this research. The title of the first study is: “Evaluating Design Decision Making via Interviewing”. The title of the second study is: “Evaluating Design Decision Making via On-looker Observations”. The title of the third study is: “Evaluating Design Decision Making via Participatory Observations”. The studies were conducted in this order.

I conducted three studies for two reasons. The first reason was to address methodological triangulation [Patton02, p 247]. I provided three different methods to gathering and analyzing data which, when combined, strengthen the overall study [Patton02, p 247]. The second reason was to be involved in the examined topic at varying levels of intimacy. This provided different viewpoints on the topic, from a single researcher. All three studies directly contribute to the first purpose of this research: they all describe software design decision making. The second and third studies directly contribute to the second purpose of this research: they both direct improvements in software design decision making. The third study directly contributes to the third purpose of this research: it highlights challenges in effecting change in design decision making.

This chapter describes the purpose of each study, and the assumptions and definitions of scope relevant to all three studies. This chapter then provides an overview of all three studies and a discussion of an empirical paradigm.

3.1 Purpose of the Studies

The primary purpose of each study was to describe design decision making through a given lens (i.e. from a given perspective), thereby contributing to answering the larger research question, How do software designers make design decisions? Aspects of qualitative studies recommend conducting a study with as few preconceptions as possible, to minimize researcher bias [Patton02]. Some even recommend not to conduct a literature review [Patton02]. However, the practicality of this is challenging – I have previous research experience that will naturally and nearly uncontrollably impact new research situations.

Thus, I made at least some of my preconceptions explicit by defining the perspective through which I examined software design decision making. I say I had, “a lens on” [Brehmer88] an existing description of decision making, and compare it to what was found in each study.

In the first study the lens used was a comparison of rational decision making and naturalistic decision making. Properties of multi-attribute decision theory (a rational approach to decision making) [Luce58, Lipshitz93b] were compared to properties of recognition-primed decision theory (a naturalistic approach to decision making) [Klein98]. The purpose of the first study was to describe design decision making through the presence or absence of the components of rational and naturalistic decision making. In the second study the lens used was a description of real options theory [Sullivan99, Boehm00, Lasher06]. Real options theory has been prescribed for solving software design decisions [Sullivan99, Erdogmus03, Boehm00]. The purpose of the second study was to describe design decision making through the presence or absence of the components of real options theory. In the third study the lens used was a description of explanation based decision making [Penning93]. The purpose of the third study was two fold: first, it was to describe design decision making through the components of explanation based decision making; second, it was to improve the use of explanation building. This last purpose aside, the intent of each study was to *describe* design decision making via a lens found in the literature on decision making.

Admittedly, at the beginning of this research it was not my intent to change lenses. Rational decision making and naturalistic decision making *seemed* sufficient enough to describe design decision making. It was not until after the first empirical study that I learned these two decision making approaches were not sufficient to describe design decision making, and I looked back to the literature for other lenses. In this way my research strategy was dynamic [Mintzberg92]. I compared design decision making to real options analysis because of existing literature recommending the use of real options analysis in software design decisions [Sullivan99, Erdogmus03, Boehm00]. Real options analysis was recommended for design decisions and I wanted to see how it compared to the design

decisions in my data. I compared design decision making to explanation based decision making because the results of the on-looker observations suggested this decision making approach might suit software design decision making. The use of rational decision making and naturalistic decision making was the intended strategy, the use of real options analysis and the use of explanation-based decision making were emergent strategies and the result was the realized strategy [Mintzberg92].

3.2 Assumptions & Definitions

Two assumptions were relevant to all three studies. The first assumption was that a software design change involves one or more design decisions. The second assumption is that a design change is a critical incident, as per [Flanagan54].

> By an incident [it] is meant any observable human activity that is sufficiently complete in itself to permit inferences and predictions to be made about the person performing the act. To be critical, an incident must occur in a situation where the purpose or intent of the act seems fairly clear to the observer and where its consequences are sufficiently definite to leave little doubt concerning its effects.

Given this definition, I emphasize that it is the design change that is a critical incident (as opposed to the entire design process), because the intent of the design change seems fairly clear to the designer making the design change.

For all three studies I use a definition of mental model taken from Johnson-Laird, and develop a definition of a strong mental model and open mental model based on this definition [Johnson05, Johnson05b]. A mental model is a model that “represents entities and persons, events and processes, and the operations of complex systems,” by satisfying the principle of iconicity, the principle of possibilities and the principle of truth [Johnson05, p187]. (Please see Appendix B, under mental model, for definitions of these principles).

A strong mental model is a mental model supported by constraints, assumptions, premises and even general ideas that are well-defined by the person who owns the mental model. In software design an example of a strong mental model of the design problem for the system

being developed is when a designer interprets a feature request in a certain way, which is supported by a design heuristic. The interpretation of the feature request may or may not be correct, and the design heuristic may or may not apply to the design problem, but the designer uses both these items to shape the solution to a design problem. An open mental model is a mental model that is supported by constraints, assumptions, premises and general ideas that are ill-defined by the person who owns the mental model. In software design an example of an open mental model of the design problem, for the system being developed, is when a designer completely (or admittedly) does not understand a feature request and is not considering design heuristics either. A strong mental model does not directly equate to a right or good mental model and an open mental model does not directly equate to a wrong or bad mental model. Another example of a strong mental model is using a design pattern because it worked well "last time". Another example of an open mental model is not understanding if an Abstract Factory design pattern or a Factory design pattern will satisfy a new customer requirement. A software designer's use of constraints, assumptions, premises and his/her general ideas applied to a design problem strengthen a mental model. That is, they shape, or structure the problem. These constraints, assumptions, premises and general ideas are termed Problem Structures.

These definitions are used to address a software designer's understanding of a software design. Initially my definition of understanding a design problem was how well-structured a design problem is for a designer [Zannier07]. A weakness of this definition is that it does not capture the cognitive aspect of a design change well. Thus, my modification to the definition of understanding a design problem is the strength of the mental model of the *design problem for the system being developed*. This definition recognizes the cognitive aspect of design, recognizes existing literature [e.g. Adelson85] and under the definition and existing literature, implicitly recognizes the role of expertise [e.g Adelson85]. I do not parameterize *how much* the designer understands the design problem, because this would require a definition of each designer's expertise, background, education, etcetera. This research recognizes that understanding affects decision making, and recognizes differences in the prominence of problem structures and how that shaped a designer's understanding of

a design problem, but this research it does not say *how much* understanding the designers had in each case study.

This thesis identifies a small, medium and large design change and the results speak to all three types of design change. A small design change is a design change made to a well structured system, and often recognized from using the system (e.g. customer use identifies a bug and the design change is a bug fix). The designer(s) implementing the design change primarily uses the existing system as constraints on his/her problem solving and validates his/her mental simulations continuously by running and re-running the system. Because the system is well structured (and functional, to an extent), this validation occurs quickly and repetitively until the solution is found. In the cases that I observed a small design change often (but not always) emerged as a bug fix found while implementing a new feature or found by a bug report. A medium design change is a design change made to a well structured system or a relatively well structured system that is recognized from number of cues, one of which is likely to be an issue of workflow in developing the software. (E.g. development is becoming difficult in some respect and a significant refactoring is required to ease development). In the cases that I observed a medium design change often emerged as a change to the class hierarchy of a system. A large design change is a change to the underlying structure of the system, or the definition of an ill-structured problem, either of which determines the general path of the development (e.g. the selection of C# and .Net as the programming language). In the cases that I observed, a large design change emerged as a change to the architecture of a system, or a change that significantly impacted the architecture of a system. Throughout this thesis there is an implicit alignment among open mental models, large design changes and ill structured problems, and an implicit alignment among closed mental models, small design changes and well structured problems [Zannier07b].

Finally I define optimal and boundedly optimal. At an extreme, optimal is defining a concept as right, according to an objective definition of right or correct, or according to a widely-accepted subjective definition of right or correct. This second provision (i.e. widely accepted subjective definition) blurs the boundary between optimal and boundedly optimal.

At a non-extreme, optimal is defining a concept as the *most* favourable or *most* desirable, under a given set of conditions. It is this non-extreme use of the term that also overlaps with the concept of Boundedly Optimal. Boundedly Optimal is defining a concept as good or best, *within a subset of potential alternatives*, that is, relative to the environment in which the evaluation occurs.

3.3 Scope of Design Decision and Design Change

The definitions of design decision and design change remained consistent across all three studies. A design decision is the selection of an alternative among zero or more known and unknown alternatives during a design change to a software application, requiring one or more implicit and/or explicit decisions. A design change is a modification to the design of a software system. This thesis does not provide a classification of software design, design change or design decision into categories such as architecture, class hierarchy, interactions and source code, as is frequently used when discussing software systems [UML07]. This is because such a classification was not found when analyzing the data.

3.4 Scope of Empirical World

The empirical "world" examined was similar across the three studies in three ways. First, the empirical world examined consisted of small software organizations (i.e. development teams of 3 – 15 people). Second, the empirical world consisted of software designers who were familiar with agile methods. In the first study, 19 of the 25 interview participants discussed a design change that occurred in an agile environment. In the second study, all three software organizations observed exhibited traits of agility and two of these three organizations were self-proclaimed agile organizations [Zannier07]. The third study did not contain as much familiarity with agile methods. Three of ten software developers in the examined software development team had previously worked in an agile software organization. Third, the empirical world examined consisted of systems that were not safety-critical or mission-critical. The software systems largely consisted of web applications, internal applications for assisting a software team's workflow, and the like. The participants interviewed and observed did not discuss safety-critical systems. Given the background of the participants of the three studies and the size of the development teams in

which this research occurred, the theory developed in this thesis concerns small software organizations, with an emphasis on agile methods and non-safety-critical systems.

3.5 Overview of the Studies

Each of the three studies can be summarized by their philosophical underpinnings, their definition of a case, their sample size, the approach to data collection and data analysis, and the approach to validity, highlighting the similarities and differences of the three studies. Such a summary is provided in Table 3.1. The first empirical study used an approach rooted in controlled experimentation to examine descriptions of design change provided by software designers. The interviews were semi-structured and the content analysis of interview transcripts was rigorous. The second empirical study used an approach that combined positivism and interpretivism. Case study summaries were developed within an initial understanding of an organization's culture. The study used on-looker observations and an approach to analysis that involved proposition building more so than content analysis. The third empirical study was entirely interpretivist, using an ethnographic account of participating in design decision making and trying to effect change in decision making. The details of the concepts mentioned in Table 3.1 are provided in Chapters 4 – 9.

3.6 Straddling the Paradigm Debate

There was a paradigm debate between quantitative and qualitative empiricism, which occurred throughout this thesis. The current state of empirical software engineering calls for scientific rigour (i.e.[Juristo01]), showing roots primarily in quantitative empiricism, with some recognition of qualitative empiricism, but little reconciliation of the two, despite such reconciliation in a highly related field of Information Systems [Mingers01, Orlikowski91]. However, the dominance of quantitative empiricism in empirical software engineering is already changing with increased popularity of examining the social side of software engineering, [Dittrich07]. All three empirical studies straddled this paradigm debate in execution or by reporting highly qualitative work to an audience predominantly accustomed to quantitative empiricism (e.g. empirical study #3). The headings of the columns in Table 3.1 blur concepts founded in quantitative and qualitative empiricism (e.g. case and sample size) as an example of the different perspectives in this research.

Table 3.1: Overview of Three Empirical Studies

	Empirical Study #1	Empirical Study #2	Empirical Study #3
Philosophical Underpinnings	Case Study	Case Study	Qualitative, Auto-Ethnography.
Definition of Case	Description of design change from interviewed software designers.	Observed occurrence of a design change in a small software organization.	6 week experience discussing and executing design change.
Sample Size	17 Design Change Examples from a developer + 8 General Design Change Discussions from a mentor **= 25 Interviews**	12 Changes at Company A + 11 Changes at Company B + 5 Changes at Company C = **28 Design Changes,** 3 Organizations	6 Design Changes containing design changes inside of them during a 6-week experience at E-Solutions Inc. **= 1 rich description**
Lens On	Rational Decision Making and Natural Decision Making	Real Options Theory	Explanation Based Decision Making
Approach to Data Collection	Semi-Structured Interviews using Critical Decision Method for Eliciting Knowledge	On-looker Observations with Question and Answer Periods	Participatory Observations & Action Research
Approach to Data Analysis	Quantitative Content Analysis using Small Window Coding and Large Window Coding. Qualitative Content Analysis using Relational Coding. Proposition forming contributing to Explanation Building. Case Study Summaries	Qualitative Content Analysis using Large Window Coding. Proposition forming contributing to Explanation Building. Case Study Summaries	Auto-ethnographic account of decision making. Auto-ethnographic account of impacting decision making.
Approach to Validity	Construct, Internal, External, Threats to Validity	Construct, Internal, External, Reliability using Inter-Rater Evaluation.	Intersubjectivity, Discussion of Authority and Representation

3.7 Development of Conclusions

Throughout chapters four to nine I present evidence contributing to the two main conclusions of this thesis. These conclusions first emerged from the data during the first empirical study and took shape during the second and third empirical studies.

The first main conclusion is that soft software designers appropriate a solution to a design decision approximately as frequently as they strive for an optimal or boundedly optimal design solution. That is, they apply a design solution that "just fits" or is "suitable". This result first emerged by trying to position software design decision making between rational and naturalistic decision making. The comparison of optimizing versus satisficing became important as the interview data showed designers discussing "good", "bad", "right" and "wrong" designs, and sometimes "good enough" design. I left the first study understanding that designers used "good", "bad" or "good enough" often, when describing design and selecting design alternatives. The data from the second study confirmed this idea when I observed designers admitting that they were just "getting the job done", without reference to some optimal solution. Finally, I participated in decisions where we just needed to make a decision, in the third empirical study. From this, the first conclusion originally was, software designers do not consistently strive for an optimal design solution. However, this did not capture what software designers *did* do. Thus, I used the word appropriate, to describe designers that employs a "good", "bad" or "good enough" approach to design.

The second main conclusion is that the approach a software designer takes in decision making is affected by the software designer's understanding of the design problem. More specifically, the stronger a software designer's mental model of a design is, the more apt s/he is to use serial evaluation (i.e. not consider alternatives to a decision problem). The more open a software designer's mental model of a design is, the more apt s/he is to use consequential choice (i.e. consider alternatives to a decision problem). This result emerged from the first empirical study and the concept of problem structuring [Guindon90a]. In [Zannier07b] I show how the data suggested that the more structured a problem, the less alternatives were considered. I left the first empirical study with this notion, but in observing software designers at work during the second empirical study, this notion was

not complete. In particular, case study 2.10B showed a software designers understanding of a design problem affected the approach to solving the problem. Thus, as will be described in Section 5.1.5, I incorporated the concept of a mental model into the second conclusion.

In these ways the conclusions were rooted in concepts taken from the literature, emerged from the initial empirical study, and were honed during the second and third study.

3.8 Chapter Summary

This thesis describes three empirical studies that contribute to one larger study addressing the question, “How do software designers make design decisions?” Each of the studies was similar in purpose and was executed under the same assumptions and definitions of scope. The studies contain descriptive case studies that are evidence for a generative model of design decision making. Given this background of the studies I now describe each study (and the components of Table 3.1) in detail, from chapter 4 to chapter 9.

CHAPTER FOUR: PREPARATION FOR AN EMPIRICAL STUDY of DESIGN DECISION MAKING via INTERVIEWING

The first empirical study consisted of conducting interviews via the critical incident technique, and analyzing interview transcripts via content analysis and explanatory case studies. This multi-case study consisted of 25 interviews with "real world" software designers. The interviews followed a semi-structured format and lasted, on average, 45 minutes. Content analysis was used to code words, phrases, sentences and paragraphs [Kripend80]. The results were compiled to build an explanation about design decision making, as it relates to rational and naturalistic decision making [Yin02, Luce58, Klein98]. This chapter describes the data collection that occurred via descriptions of the approach to the interviews descriptions of the interview participants. This chapter then describes the data analysis that occurred. Last, this chapter describes my experiences with qualitative inquiry after this first empirical study. In this study, a "case" or "unit of analysis" is one description of a design change from a software designer.

4.1 Method of Data Collection

The Critical Decision Method for Eliciting Knowledge (CDM) was the foundation for the format of the interviews [Flanagan54, Klein89]. The CDM begins with one general question to initiate conversation with the interview participant and to hear a description of the critical incident. The CDM then provides probing questions based on the information given in the description. It is a semi-structured interview format, meaning the CDM was used as a guide [Patton02, p349].

In the interviews, I asked designers to describe a design change s/he had made, and followed this question with probing questions addressing aspects of his/her response to the first question. The order and the way in which the questions were asked varied, depending upon the interview participants' responses, but the theme of the question remained the same. Examples of the way the questions were asked are listed in Table 4.1. An example of a variation of a question is, for external goals, "You mentioned your marketing department

wanted you to use XSLT, can you talk about that a bit more?" I chose the CDM as an interview guide because of its extensive use in analyzing decisions [Klein89, Klein98].

Table 4.1 Critical Decision Interview Example Questions [Klein89]

Decision (initial question) Describe how you make a design change to a system, and how you make the decision to make the change.
Cues (probe) What do you see, hear, discuss, or experience that suggests a change needs to occur?
Knowledge (probe) Where do you acquire the knowledge to make the change?
Options (probe) Discuss the extent to which you consider options in making a design change to a system.
Experience (probe) To what extent do specific past experiences impact your decision to make a design change?
Time Pressure (probe) How does time pressure impact decisions in design changes?
Externals (probe) How do external goals impact decisions in design changes?

4.1.1 Interview Participants

The participants interviewed were software developers and software mentors with varying degrees of experience in software development. To contact potential interview participants I used snowball sampling, an emergent sampling approach that does not *explicitly* characterize an interview participant by domain, type of system, experience, or any other characteristic [Patton02]. In snowball sampling, one interview participant leads to another interview participant via an introduction or a comment such as, "You should talk to…"

I initially interviewed colleagues at the University of Calgary who had previous software design experience, and members of the local software industry. This meant that initially the interview participants were primarily Calgary-based software developers (11 interview participants). When a leading agile conference was held in Calgary in August of 2004 [Agile04] the demographic of the interview participant broadened. I interviewed leaders of the agile community and developers from across North America. I interviewed 25 people in total.

While interviewing, I noticed a difference in the perspectives of 8 of the interview participants. These subjects discussed design changes they had seen people make or had overseen when they were a coach or a consultant on a team. These subjects spoke about design change as a concept rather than one critical incident, and gave multiple examples of design changes. Given this, I defined two perspectives of interview participants: developer and mentor. A developer is an interview participant who discussed a design change that they championed when s/he was a member of a development team. A mentor is an interview participant who discussed design change as a concept, based on the culmination of design experiences with software development teams where s/he was a paid coach or a consultant. When referring to both mentors and developers I used the word Designers.

4.2 Method of Analysis

Analysis occurred via a three-step process. The first step was the use of content analysis; the second step was building explanatory case studies; the third step was performing cross-case analysis. The interviews were transcribed and analysis occurred on the transcripts of the interviews.

4.2.1 Content Analysis

Content analysis places words, phrases, sentences or paragraphs into codes which can be pre-defined or interactively defined [Kripend80]. I created an interactive dictionary by reading five random interviews from the larger pool of interviews, and grouping similar words and phrases together. I developed 42 codes and added codes as I continued to read interviews [Kripend80]. After a new code was added I did not go back and re-code previously coded interviews as this was simply too time consuming [Kripend80]. The 42 codes that emerged had conceptual similarities and as a result I grouped the codes into 6 categories. These categories and the groupings are provided in Appendix B. Primarily, the categories helped me understand the vast amount of information through which I was sifting.

I coded at a very detailed level first, coding words and phrases only. I coded at a broader level next, coding sentences and paragraphs. I then coded the general relationships between codes and specific relationships between codes in answer to an interview question. These four types of analysis were performed on all the interviews. Each analysis is built on the previous analysis. They are described below.

Small Window Coding: In small window coding (SWC) I tagged one word or one phrase at a time. SWC potentially removes the context of an interview but shows the word choice of an interview participant [Kripend80]. This approach shows no relationships between codes but is best used to show frequencies of codes. A picture of the real coding I performed is shown in Figure 3.1

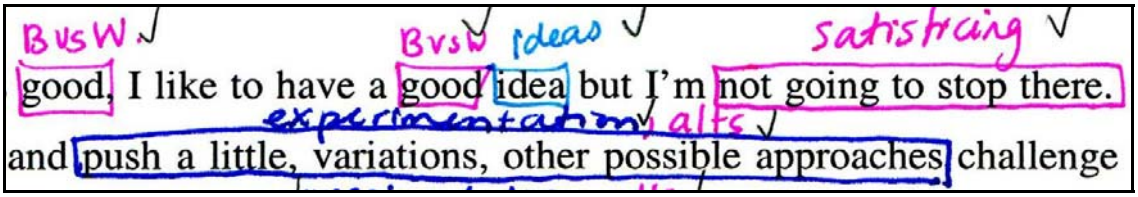

Figure 3.1 Small Window Coding, Scanned Photo of Actual Coding

Large Window Coding: In Large Window Coding (LWC) I tagged one phrase, full sentence or paragraph at a time. LWC incorporates some context of a statement and shows word choice in the context of an interview participant's responses. This approach shows no relationships between codes but is best used to show frequencies of codes. A picture of the real coding I performed is shown in Figure 3.2. The text that was coded was highlighted, and in Figure 3.2 you can see the beginnings of the relationship coding that I conducted. I also tried to assign a weight and a sign to the relationships between codes, but this became too complex to maintain over the long term.

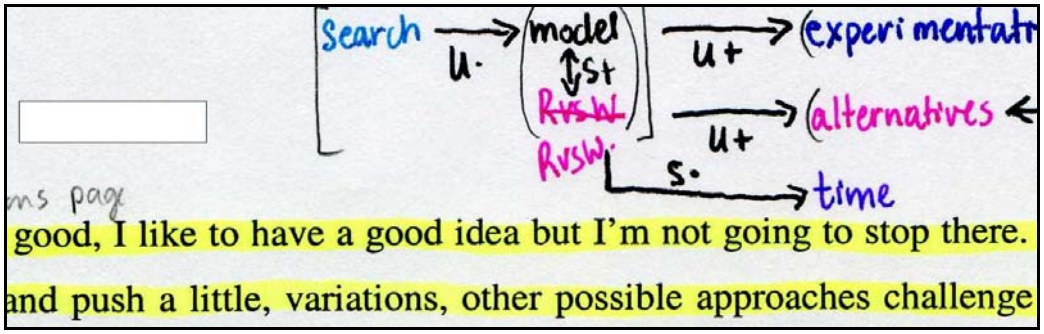

Figure 3.2 Large Window Coding, Scanned Photo of Actual Coding

Generic Relational Coding: In Generic Relational Coding (GRC) I tagged relationships among codes. GRC does not answer a specific question but incorporates much context of a statement. It is best used to show recurring themes. Figure 3.3 shows the relational coding. I searched for relationships such as **Model – Right vs. Wrong**, with no reference to the question that was being asked.

Specific Relational Coding: In Specific Relational Coding (SRC) I tagged relationships among codes in answer to a specific question. SRC incorporates much context of a statement is best used to show recurring themes in answer to a question. Figure 3.3 shows the relational coding. I searched for relationships such as **Model – Right vs. Wrong**, with reference to the question that was being asked, and this is shown in Figure 3.3. with the comment "what he does" in the left margin. This was in response to the question on Options.

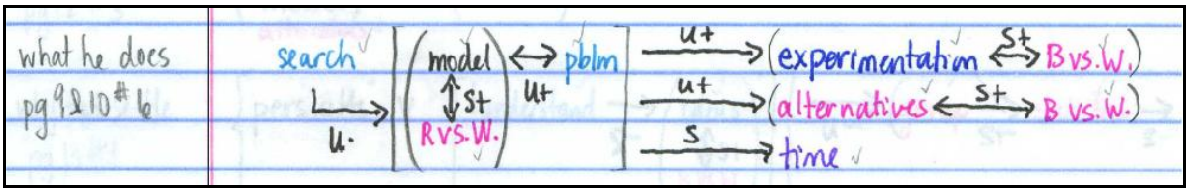

Figure 3.3 Relational Coding, Scanned Photo of Actual Coding

I used content analysis because it offered a systematic and reliable approach to condensing text into codes that can be analyzed. All four types of coding were inductive and were used to validate results generated from explanation building.

4.2.2 Explanatory Case Studies

After performing the content analysis I examined each interview as an explanatory case study of the context and circumstances surrounding one or more design decisions. The case study summaries can be found in Appendix A. I used the four types of coding to build a summary of each case study. I then coded this summary and compared this coding to the previous inductive coding I had performed, for validation. A match in the comparison meant that I had coded an interview question the same way twice. For all interview

participants I achieved at least 5 matches out of the 7 interview questions. A more detailed discussion of matches between the two sets of codes can be found in Chapter 11.

I then interpreted each question summarized in the case study as it related to the attributes of rational decision making (RDM) and naturalistic decision making (NDM). Differences existed between RDM and NDM in four respects: in the goal of the decision, the method of the decision, the effect of environment on the decision and the nature of the knowledge employed in the decision. These points are summarized in Table 4.2.

Table 4.2: Interpretations of Interview Questions with respect to Decision Making

Component	**1 RDM**	**2 NDM**
Decision Goal	(1.1) *Optimizing:* Cues are right or wrong, quantifiable	(2.1*) Satisficing*: Cues are better or worse, not quantifiable.
Decision Method	(1.2) *Consequential choice*: Options are considered.	(2.2) *Singular Evaluation*: Options are not considered.
Decision Environment	(1.3) *Not concerned with computation overhead*: Time pressure is not a factor in decision making. External goals do not impact decision making. Cues are quantifiable.	(2.3) *Dynamic conditions*: External goals impact decision making. Time pressure impacts decision making. (2.4) *Real-time reactions*: Time pressure is an issue. Cues are from some trigger. (2.5) *Ill-defined tasks & goals*: Externals impact a decision. (2.6) *Situation assessment*: Cues are unquantifiable.
Decision Knowledge	(1.4) *Cognizant of all possible courses of action*: Specific experience based knowledge, explicit search of knowledge.	(2.7) *Tacit based knowledge*: Accumulation of knowledge. (2.8) *Experience-based knowledge*: Accumulation of experience.

Interpretation 1. If a designer discussed cues (that a design change needed to occur) as being right or wrong, the designer's goal was to optimize the design, and the goal of the decision was rooted in rational decision making. If the designer discussed cues as being better or worse, the designer's goal was to satisfice the design, and was rooted in naturalistic decision making.

Interpretation 2. If the designer discussed numerous alternatives surrounding the design change then s/he used consequential choice, which is rooted in rational decision making. If the designer did not discuss alternatives, but discussed a single solution that worked, then s/he used serial evaluation, which is rooted in naturalistic decision making.

Interpretation 3. If a designer was unconcerned with time pressure and the external environment then s/he was unconcerned with computational overhead and external goals, which is rooted in rational inquiry. If a designer was concerned with time pressure, changing conditions, and ill-defined goals then and there was little time and point in considering detailed computations and external goals influenced decision making, which is rooted in naturalistic decision making.

Interpretation 4. If the decision maker used explicit searches, explicit references to knowledge (e.g. in textbooks) or explicit references to past experience then the designer was cognizant of all possible courses of action, which is rooted in rational decision making. If the designer relied on a general accumulation of experience and knowledge and did not perform explicit searches for knowledge then s/he was not cognizant of all possible courses of action, which is rooted in naturalistic decision making.

These interpretations were used to determine if each participant followed decision approaches that aligned with rational decision making or naturalistic decision making.

4.2.3 Cross Case Comparison

The third step of analysis was to compare the results between cases. I made statements about design decisions for an initial case, then revisited the initial case and examined new cases to continuously shape the statements made [Yin02]. I continuously compared case studies to build the explanation of design decision making relative to rational decision making and naturalistic decision making.

4.3 Experiences with Qualitative Inquiry

This empirical study was my first experience with qualitative research. With the advantage of hindsight I can identify 5 strengths and 4 weaknesses of the study. First, for the strengths, I had an interview guide (the CDM) that had been well-used in interviews. Second, the setting for all the interviews was the most convenient location for the interview participant, ensuring s/he was as comfortable as possible. Third, I performed three practice interviews, before conducting the majority of my interviews. Fourth and fifth, the interview participants were not coerced to participate in the interview and gained or lost very little through their participation – thus reducing the chance of any hidden agendas in an interview participant's responses. For the weaknesses, it was first interesting to note how stereotypically confused I was at times [Patton02]. There were moments where I believed I had more data than could possibly be organized and moments where I was certain the data I had were completely useless. In short, I experienced all the usual aspects of performing a qualitative study [Patton02]. During the interviews I had the potential to ask leading questions. This occurred at least once in the data I transcribed and I removed it from the analysis. Third, the study was subject to the memory of the interview participant. Interview participants explicitly stated when s/he didn't remember enough about a design change to answer part of a question. Fourth, the interviews were subject to an interviewee's desire to please [Patton02]. I saw signs of a desire to be a "good" interview participant in 2-3 of the subjects. This was apparent when interview participants said a version of the phrase "I'm not sure if that is the sort of answer you're looking for." To this I typically responded with a version of the phrase, "I am looking for anything you can recall surrounding the design change.".

4.4 Chapter Summary

The first empirical study used interviews with "real world" industry participants to understand what aspects of rational decision making and naturalistic decision making occur in software design decision making. Twenty-five interviews were conducted. Analysis involved four types of content analysis, explanation building using developed interpretations, and cross case analysis. I now present the results of this first empirical study.

CHAPTER FIVE: RESULTS OF AN EMPIRICAL STUDY of DESIGN DECISION MAKING via INTERVIEWING

Results from interviews with developers and mentors yielded frequencies of codes from content analysis, recurring relationships between these codes, themes related to rational and naturalistic decision making and general responses to interview questions. These four components are evidence for the two main conclusions and set of eight smaller results, presented throughout this thesis. The first main conclusion is that designers appropriate a design solution approximately as often as they strive for an optimal design solution or a boundedly optimal design solution. The second main conclusion is that the strength of a designer's mental model impacts the extent to which that designer considers alternatives. The smaller results found in this study address time pressure, external goals and active searches for knowledge. This chapter addresses each of these results using the four types of data mentioned above.

Throughout this chapter I use identifiers such as *CS1.1_Q1* to tag an excerpt of an interview. The format of this identifier is as follows: The "CS" stands for "Case Study". The 1.1 means the excerpt comes from the first empirical study and the first interview participant. (1.2 is from the first empirical study and the second interview participant). The "Q1" (or "Q2") represents the first quote (or second quote) taken from the interview. The quotes are ordered for organizational purposes only.

5.1 Optimal Design Solutions?

The first conclusion that emerged from the data is that software designers appropriate a design solution approximately as often as they strive for an optimal design solution or a boundedly optimal design solution. I will show the 4 ways in which this conclusion emerged from the data: frequencies of codes, recurring relationships between codes, and general responses from two interview questions and relevant themes from rational decision making and naturalistic decision making.

5.1.1 Frequencies of Codes

The frequencies of codes generated from Small Window Coding (SWC) and Large Window Coding (LWC) were intended to highlight potentially important concepts in the interviews. While the majority of the results came from relationships between codes, interpretations applied to quotes and responses to questions, the frequencies were a starting point to the results. To determine the role of pursing an optimal or boundedly optimal design solution, I present frequencies of the codes **Right vs. Wrong, Better vs. Worse**, **Satisfice** and **Weight**. I compare frequencies of the codes **Right vs. Wrong,** and **Better vs. Worse,** and frequencies of the codes **Satisfice** and **Weight.**

I have stated that I did not go back to re-code when a new code was added. This could potentially impact the frequencies that I present in the following pages. However, the codes that I added were not prominent in the coding process. For example, the code **Time** was added to handle words and phrases such as "now" or "at this time". However, this code did not emerge as dominant in the entire coding process. Often such a code occurred close to 0% in the frequencies. Many other concepts and codes (e.g. **Right vs. Wrong, Better vs. Worse**, **Model**, **Knowledge)** emerged more prominently and captured concepts discussed throughout this thesis. It is for this reason that the frequencies presented here are still worthy of examination.

First, I present the number of interview participants who had the codes **Better vs. Worse** and **Right vs. Wrong** in their top three codes, for SWC and LWC. As shown in Table 5.1, **Better vs. Worse** appeared in the top 3 codes of 5 more interview participants than for **Right vs. Wrong**. More specifically, **Right vs. Wrong** was in the top three codes for *no* interview participants in SWC, and for only one interview participant for LWC.

Table 5.1: Number of Interview participants with Code in 3 Most Frequent Codes

Window Size	Small Window Coding		Large Window Coding	
Code	Better vs. Worse	Right vs. Wrong	Better vs. Worse	Right vs. Wrong
Number of Interview participants	5	0	6	1

To speak to the sensitivity of these numbers, the code **Better vs. Worse** appeared in the top 10 codes of 11 more interview participants than for **Right vs. Wrong** in SWC, and it appeared in the top 10 codes of 4 more interview participants than for **Right vs. Wrong** in LWC. The numbers are shown in Table 5.2.

Table 5.2: Number of Interview participants with Code in 10 Most Frequent Codes

Window Size	Small Window Coding		Large Window Coding	
Code	Better vs. Worse	Right vs. Wrong	Better vs. Worse	Right vs. Wrong
Number of Interview participants	13	2	10	6

Second, I present the frequency of **Better vs. Worse** and of **Right vs. Wrong** for SWC and LWC for all interview participants. The frequency is presented as a percentage of the total number of codes for each interview participant (n.b. the total number of codes varied for each interview participant). Table 5.3 shows these frequencies. Only 3 of the 25 interview participants show a frequency of **Right vs. Wrong** as greater than or equal to the frequency of **Better vs. Worse,** for both of SWC or LWC. That is, in 22 of 25 interviews (88%) words that could be coded as **Better vs. Worse** were found more often than words that could be coded as **Right vs. Wrong**, regardless of the window size (SWC or LWC).

From these two points the concept of **Better vs. Worse** is more prevalent than the concept of **Right vs. Wrong** when discussing a design change. From the data, I state that, the frequency of a designer discussing a right (or wrong) aspect of a design change was smaller than the frequency of a designer discussing a better (or worse) aspect of a design change. From the code definitions I state that, when discussing concepts as being **Better or Worse** one uses subjective qualifiers, and when discussing concepts as being **Right or Wrong** one uses objective qualifiers. Therefore, the frequency of a designer discussing an objectively right (or wrong) aspect of a design change was smaller than the frequency of a designer discussing a subjectively better (or worse) aspect of a design change. By the definition of bounded optimality and from these points I state that a subjectively better (or worse) aspect

of a design change is, at best, boundedly optimal. Thus, my first argument is that, at best, software designers strive for boundedly optimal software design solutions.

Table 5.3: Comparing Better vs. Worse and Right vs. Wrong Frequencies in Interview Participants

Participant ID	Small Window Coding (%)		Large Window Coding (%)	
	Better vs. Worse	Right vs. Wrong	Better vs. Worse	Right vs. Wrong
1	7.6	1.8	3.6	2
2	5.5	1.5	3.3	2.5
3	6.9	2.2	11.5	1.6
4	1.2	0	2.5	0
5	4.8	2.4	5.7	2.9
6	3.6	2.1	5.2	3.6
7	5.8	2.0	9.2	4.8
8	3.1	.70	4.5	1.5
9	3.5	2.5	6.3	7.0
10	2.0	.14	1.7	0
11	1.7	.77	1.2	0
12	3.4	.30	4.3	.94
13	4.4	2.0	3.2	2.6
14	5.8	3.3	9.1	2.3
15	2.2	1.3	2.9	1.6
16	1.9	.30	1.1	1.1
17	4.2	.44	2.8	.60
18	2.1	.40	1.2	.60
19	3.1	.31	2.5	.50
20	1.6	2.6	0	4.3
21	5.3	.53	7.7	0
22	1.8	.89	1.8	0
23	1.8	3.3	2.3	5.1
24	4.9	3.2	3.3	3.3
25	2.2	3.0	2.1	2.1

Given this argument, I now present results that address the bounds placed on designers' boundedly optimal design solutions, assuming that they do strive for boundedly optimal solutions. To do this I present the results of the codes **Satisfice** and **Weight**. When looking at the list of codes found in Appendix B, it is clear that there is no code called Optimal, or the like. The dictionary of codes was interactive which means use of words that could be coded as **Optimal** did not appear to the extent that there was merit developing a code. However, the code **Weight** did emerge. Since this code suggests the comparison of advantages and disadvantages, leading (ideally) to a boundedly optimal solution, I

compared the frequency of the code **Satisfice** to **Weight** in the same manner as I compared the code **Better vs. Worse** to **Right vs. Wrong.**

First, neither the **Satisficing** code or the **Weight** code appeared in the top 3 most frequently occurring codes, for any interview participant. I present the frequency of **Satisfice** and **Weight** for SWC and LWC, for all interview participants. Again, the frequency is presented as a percentage of the total number of codes for each interview participant. Table 5.4 shows the frequencies. Twelve of the twenty-five participants show a frequency of **Weight** as greater than or equal to the frequency of **Satisfice**, for both SWC and LWC. In other words, just over half (13/25) of the interview participants discussed satisficing more than discussing the use of weights when discussing design change.

Table 5.4: Comparing Satisfice and Weight Frequencies for all Interview Participants

	Small Window Coding (%)		Large Window Coding (%)	
Participant ID	Satisfice	Weight	Satisfice	Weight
1	1.9	1.3	3.2	0
2	1.5	2.0	0	2.5
3	1.1	.65	.82	2.5
4	.49	.24	0	.83
5	.59	.88	2.0	.82
6	.43	1.3	1.6	2.38
7	1.4	.77	.44	.88
8	1.7	.56	.75	.75
9	.87	.63	0	1.4
10	1.3	.99	1.7	0
11	.71	1.6	.62	1.2
12	.89	.97	1.4	0
13	.87	0	1.3	0
14	.62	.41	.57	0
15	1.1	.42	.41	0
16	0	.81	0	0
17	.11	.22	0	0
18	.18	.96	.60	1.2
19	.08	1.4	0	.50
20	.23	.23	0	1.1
21	0	0	0	0
22	0	.89	0	1.9
23	.41	.98	0	2.3
24	.64	.58	.82	.82
25	.08	.68	.69	.69

From these two points the concept of satisficing is at least as prevalent as the concept of weights, in the interviews. From the data I state that a designer discussed satisficing when discussing aspects of a design change at least as often as a designer discussed weighing the good and bad aspects of a design change. Given my first argument, that at best designers strive for boundedly optimal design solutions, and given that satisficing occurred at least as often as weighing alternatives, I argue that software designers do not consistently strive for an optimal or boundedly optimal design solution when discussing design changes they have made. This argument is made using the frequencies of codes. I continue to support this argument through emerging relationships found in the interview transcripts.

5.1.2 Emerging Relationships

The Generic Relational Coding (GRC) exposed relationships between codes representing components of design (e.g. **Model**, **Code**) and the code **Better vs. Worse**. These relationships were used to understand the words designers used to discuss aspects of design. Below are quotes showing relationships between **Better vs. Worse** and **Code**, **Model**, **Code**, **Recognition** and **Customer Product**. These quotes are evidence for the argument that, at best, designers strive for a boundedly optimal design solution.

When asked where designers develop the knowledge used in a design change, this mentor prioritized colleagues and experience over books.

CS1.1_Q1, **Better vs. Worse – Code***: "There is nothing like working with good examples… working with good code, other people's good code, that is really great to work with… and you learn of course by interacting with good programmers and good designers and good whatever it is you're working with." Mentor Perspective*

When asked to describe his experience with design change this mentor discussed incremental change.

CS1.5_Q1, **Better vs. Worse – Model***: "…you're making not these grand decisions, but you're making little tiny atomic changes that are sort of making the design better locally. You've got all these local things and it's globally better." Mentor Perspective*

This developer was asked if he would have changed anything in the design change he described, but had no regrets.

CS1.3_Q1, **Better vs. Worse – Recognition***: "Knowing what we needed to do and the ultimate direction we were going in, and knowing the problems we were facing, those 2 things, where we were going and what problems we were facing today, [the design change] seemed like a very good fit." Developer Perspective*

When asked how past experiences were re-used this mentor was careful to distinguish projects.

CS1.2_Q1, **Better vs. Worse – Customer Product***: "But just because something succeeded on one project doesn't necessarily mean it's the right solution on the next project. It doesn't mean you should reject it, but it also doesn't mean you say 'ok, we'll do this on the next project because it worked on the last project.' I think that's the danger of, for example, certification and standardization, is that project are different, all projects are different." Mentor Perspective*

These quotes show an emphasis on design solutions that are qualified as better or worse, not right or wrong. The below quotes show how the code **Right vs. Wrong** related to the code **Model** and **Personal Attributes**.

This mentor was asked where he acquired knowledge to make a design change and while he addressed the idea of a "right" design solution, he did not discuss experience in achieving a "right" design solution. Instead he discussed a gut feeling about right or wrong.

CS1.5_Q2, **Right vs. Wrong – Model***: "…often when mortals like you and I discover a pattern we don't get it quite right. When you read the write up that the Gang of Four did on the patterns, it's like they have them all crisp and cool and that's just right. But I maybe never got there. Maybe I never had that. I got something in that direction. 'I did a thing like that once,' you say. Mentor Perspective*

CS1.5_Q3, **Right vs. Wrong – Personal Attributes***: "I said, 'I don't feel good about what we did – it all works but it doesn't feel right. It just doesn't feel the way it's supposed*

to feel when we do this stuff, and yeah, it's working but there's something in there that just isn't right." Mentor Perspective

These quotes indicate the manner in which designers discussed the design of software projects. The emphasis on **Better vs. Worse** over **Right vs. Wrong** emerged from the data and produced the point that design was discussed in a subjective fashion. Again, the argument is that subjective discussions of design imply, at most, boundedly optimal design solutions. The comparison of Satisficing and Weights is addressed in Sections 5.1.3 and 5.1.4.

5.1.3 Responses to Questions

Quotes highlighting the use of satisficing emerged from the analysis, contributing to the point that designers do not consistently strive for an optimal or a boundedly optimal design solution. The quotes provided below were all coded, in some part, using the code **Satisfice**. These quotes are part of responses to the interview questions and the specific question is listed before the quote (i.e. the theme name as per Table 4.1).

This developer discussed applying XML and XSLT to an existing design to clean up code.
CS1.3_Q2, **Decision***: "… at that point it seemed as if [XSLT] had become mature enough to use." Developer Perspective*

In the course of discussing how he recognized a design change needed to occur this mentor discussed providing small pieces of functionality in order to ship code early.
CS1.5_Q4, **Cues***: "The thing that happens in an agile design is that you have just enough design for the moment and you're trusting that you will be able to elaborate that design…" Mentor Perspective*

This mentor described a project where he was coaching a team to use agile practices, while still accommodating the team's organizational need for documentation.
CS1.6_Q1, **Decision***: "…satisfy the need for documentation with really no deviation from the adaptive, evolutionary design they were doing anyways." Mentor Perspective*

When asked a follow-up question to his description of design change this mentor discussed when a team was "done" with design and how much documentation was required.
CS1.6_Q2, **Cues***: "Teams that have worked for a year on the same code base, the design is just there. It might not be on paper, it might not even be formally documented but [the team] know[s]. It's there. They can talk in a very abbreviated way, they can sketch something nobody could read but them. But it's sufficient and they'll get it implemented." Mentor Perspective*

This mentor discussed how quickly he rejected alternatives, suggesting that at times it could be quickly.
CS1.7_Q1 **Options***: "…if [an option] is not obviously a bad thing, then its probably good enough for my purposes for now." Developer Perspective*

This developer explained the details of using XSLT in his design change.
CS1.8_Q1, **Options***: "XSLT isn't exactly fast to begin with but it was fast enough for us." Developer Perspective*

This developer discussed the use of XML and COM as being a design change that would be useful in future design situations.
CS1.10_Q1, **Decision***: "I tried to make [the design] general enough so that it could be used for anything but specific enough so it could definitely be used for this project that was the driving force." Developer Perspective*

Problems in the existing user interface of a software application prompted a change for this mentor.
CS1.12_Q1, **Decision***: "…our graphs weren't restrictive enough…" Developer Perspective*

This mentor then discussed a facet of iterative agile development.

CS1.12_Q2, **Knowledge***: "… you take this moment…that it's good enough to get the next test to pass and then generalize from there…" Developer Perspective*

This developer described a Refactoring to a web application where he separated business logic from presentation logic.

CS1.13_Q1, **Options***: "I was taught to code till [the code] works then clean [the code] up." Developer Perspective.*

This developer discussed the train of thought leading to making a feature of an application easier to use and better known to people.

CS1.14_Q1, **Cues***: "…we were thinking of enhancing the HTML capability but that wouldn't have been enough [to solve the problem] so we had to switch to applets…" Developer Perspective*

A technical problem in the existing application prompted a design change for this developer.

CS1.24_Q1, **Decision***: "…we didn't' have a good import structure… it wasn't going to work for what we needed." Developer Perspective*

Just getting the job done was made very clear by this developer describing a design change under time pressure.

CS1.24_Q2, **Decision***: "… it just had to work, it didn't have to have all the fancy clowns and all the bells and whistles it just really had to work." Developer Perspective*

In total 12 of the 25 interview participants referenced a "good enough", or "satisfactory" design, knowledge of a design, underlying technology, or the like. Please note, the 12 quotes above do not map directly to the 12 participants discussing satisficing as provided in Section 5.1.1. Five more quotes concerning satisficing are provided in Section 5.3, Study-Specific Results, in the discussion on knowledge. The quotes here are evidence for the point that developers do not consistently strive for optimal or boundedly optimal design solutions. Satisfactory solutions are frequently employed.

For comparison, I provide quotes found from using the code **Weight**. Ten quotes were found and are provided below.

Describing his recommendations for decision making, this mentor highlighted a challenge designers have with extensive forms of rational decision theory.
CS1.2_Q2, **External Goals***: "…people don't deal well with multiple criteria that don't have relative priority ranking. So one of the things that I like to have them do is, force people to prioritize steps so they know the two or three things that are the highest priority." Mentor Perspective*

Again, this mentor emphasized the scope in considering alternatives is smaller than extensive forms of rational decision theory.
CS1.2_Q3, **Options***: "You often don't want to keep 50 options open, but narrow those 50 options to 3 or 4, but keep those options open as long into the process as you can." Mentor Perspective*

Last, this mentor said that delaying decisions allows decision makers to learn more about design alternatives.
CS1.2_Q4, **Options***: "…come up with criteria for those decisions and then say it may be that once we come up with those options, that will give us placeholders to accumulate information." Mentor Perspective*

This mentor highlighted the bounds of extensive use of rational decision theory but did say that tradeoffs were made in iteration planning meetings.
CS1.6_Q3, **Options***: "We don't coach to do a real exhaustive analysis of the options and then weigh the options and then have this formal process that fuels a decision. That would just take too long." Mentor Perspective*

Options are weighed, according to this mentor.

CS1.6_Q4, **Options***: "I think we definitely see different options being weighed…" Mentor Perspective*

This developer clearly showed the lack of a "right" or "good" solution when comparing decision alternatives.

CS1.7_Q2, **Options***: "…the good thing is that if there are 2 remaining options that are fairly equivalently good, then it matters less which you actually pick." Developer Perspective*

This mentor discussed the issue of external factors affecting decision making and how alternatives are considered, throughout the interview.

CS1.9_Q1, **Options***: "I just see too much of that type of thing where there's other considerations, [external goals] in [the decision] that either get weighed too heavily or don't get weighted at all and should be." Mentor Perspective*

This mentor discussed inter-team competition and collaboration to consider alternatives. For example, members of development and members of marketing "lobby" members of quality on what features make it into a product release.

CS1.11_Q1, **Time Pressure***: "The team, over a period of discussions, makes [groups of people within the team] figure out how to trade-off." Mentor Perspective*

Alternatives are considered on an issue such as credit card processing, according to this developer.

CS1.21_Q1, **Options***: "We probably spent say a week in total looking at solutions and identifying the strengths and weaknesses of them before we settled on everyone agreeing…" Developer Perspective*

This developer discussed a side project he was developing where he wanted to learn.

CS1.25_Q1, **Decision***: "I wrote down every option, sort of a quick pros and cons around each one… depending upon the context in which I was solving the problem I would weigh the different criteria with regards to which one I would use." Developer Perspective*

In total 7 of the 25 interview participants referred to weighing options, or tradeoffs. There were 4 other quotes that show the priority of features relative to other features. These quotes are shown below.

People's priorities and workflow issues were an indicator that a design change could be made, according to this developer.
CS1.3_Q3, **Cues:** *"… they were focused on delivering a lot of features and not updating the UI."*

Feedback from people who interact closely with the customer was an indicator that a design change could be made, for this developer.
CS1.18_Q1, **Cues:** *"If we get 2 or 3 of the decent [product dealers, who interact with the customer] saying 'you know what, I've run into this [feature] and it sounds like a really good idea,' ok, we give [the feature] more priority I guess, or weight." Developer Perspective*

This developer discussed priority as indicating changes that could be made to the design.
CS1.19_Q1, **Time Pressure:** *"So all of the GUI changes and fixes which were really really low priority at one time, all of a sudden have been able to move up the list." Developer Perspective*

This developer discussed priority as well as a need for more developers as indicators of design changes that need to be made.
CS1.23_Q1, **Decision:** *"We often get more work thrown at us with high priority that has to supersede other work we're doing." Developer Perspective*
These quotes containing the code **Weight** and the code **Satisficing** are evidence for the argument that software designers do not consistently strive for an optimal or boundedly optimal design.

To clarify, the frequencies of **Weight** and **Satisfice** found in Section 5.1.1 are not aligned with the number of quotes provided here. A quote incorporates much more context than a frequency, which is why both sets of data are provided. These quotes show at least as much emphasis on the concept of Satisficing, as on the concept of weights.

5.1.4 Rational Decision Making, Naturalistic Decision Making

The interviews were analyzed with a lens on rational decision making and naturalistic decision making. I searched the interview transcripts for one of two points, to support or refute the idea that designers do not consistently strive for an optimal or boundedly optimal design solution. I searched for subjective cues that a design change needed to occur and I searched for objective cues that a design change needed to occur (Table 4.2, points 1.1 and 2.1). As per the initial interpretations in Table 4.2, subjective cues are an indicator of satisficing, a trait of naturalistic decision making; objective cues are an indicator of optimizing, a trait of rational decision making.

I summarized just the question on cues and classified the response as either NDM or RDM, depending on references to subjective cues or objective cues, respectively. Of the 25 interviews, 8 interview participants responded to the question on cues in a manner that could be classified as rational. Table 5.5 shows these classifications. Seventeen out of 25 (68%) responded to the question on cues in a manner that could be classified as naturalistic. Indicators of satisficing were found in 68% of the responses to cues [Zannier07b]. The use of naturalistic decision making via satisficing is evidence for the argument that software designers do not consistently strive for optimal or boundedly optimal design solutions.

Table 5.5: Classification of Response to Cues Question ('*' indicates a mentor perspective)

Interview ID		1*	2*	3	4	5*	6*	7*	8	9*	10	11*	12*	13	14	15	16	17	18	19	20	21	22	23	24	25
Cues	RDM												X	X	X			X			X	X			X	X
	NDM	X	X	X	X	X	X	X	X	X	X	X				X	X		X	X			X	X		

articulate, for 3 reasons. First, the statement does not incorporate the type of problems in which software developers work. Previous literature [Rittel70, Guindon90a, Guindon90b] shows software development as lacking an optimal solution, by definition. Given this, stating that designers do not consistently strive for an optimal design solution is *seemingly* redundant. Second, this statement does not highlight the way in which designers discuss design. Ideas of better or good design, or interesting qualifies for design such as "spaghetti code" were in abundance, showing that designers have ideas about how to do software design well and poorly. In this sense, boundedly optimal design solutions are pursued, albeit intermittently. This is important to recognize, despite the lack of a defined optimal in software development. Third, this statement suggests an emphasis on satisficing that is arguably a bit extreme. The term satisficing came from emergency situations where "good enough" was measured by human lives saved. This is arguably a more concrete measure than what "good enough" software design is. Because of the cognitive skill required in software design and because of the numerous ways in which "good" can be measured in software design, what is "good enough" for one designer is not necessarily "good enough" for another designer.

Given these reasons I formulate the first conclusion as follows: Software designers *appropriate* design solutions approximately as often as they strive for an optimal or boundedly optimal design solution. In the context of software design I define Appropriate as the application of a design idea to a design change, where the idea accommodates the problem regardless of varying levels of awareness of design alternatives that the software designer making the design change may have. The first part of this definition is taken from the definition of appropriate as per [MerriamWebster07] and the second part suggests a non-extreme use of satisficing.

5.1.6 Summary of First Conclusion via Interviews

The first empirical study used interviews as evidence for the point that designers appropriate design solutions approximately as often as they strive for an optimal or boundedly optimal design solution. This sub-section (Section 5.1) has contributed to this point in four ways:

- Through frequencies of codes, showing the code **Better vs. Worse** occurring more often than the code **Right vs. Wrong**, and the code **Satisfice** occurring at least as often as the code **Weight** (13/25 versus 12/25).
- Through relationships with the code **Better vs. Worse** occurring more often than relationships with the code **Right vs. Wrong**, and through quotes representative of the manner in which designers discussed design.
- Through specific responses to questions showing quotes coded as **Satisfice** and through 12/25 interview participants referencing a good enough or satisfactory design.
- Through classifying 68% of interview participants' response to the question on Cues, as following naturalistic decision making.

5.2 Approach to Design Decision

The second conclusion that emerged from the data is that when making a design change, the approach a software designer uses depends upon his/her understanding of the design problem. To show how this conclusion emerged from the data, I will present evidence that stronger mental models lead to the use of serial evaluation and more open mental models lead to the use of consequential choice. The evidence for this emerged in four ways: frequencies of codes, recurring relationships between codes, specific responses to interview questions, and relevant themes from rational decision making and naturalistic decision making. Each of these subsections will provide information on two points: a) the knowledge and ideas surrounding a designer's discussion of the model of the system in which the design change was being implemented, b)the use of either serial evaluation or consequential choice.

5.2.1 Frequencies

Once again, the frequencies of codes generated from Small Window Coding (SWC) and Large Window Coding (LWC) were intended to highlight potentially important concepts in the interviews, as a starting point to the qualitative analysis. For the topic of the strength of the mental model and its relationship to consequential choice, I present frequencies of the codes: **Model, System Breakdown, System Integration, Code, Knowledge, Ideas** and

Alternatives. The idea of a mental model can be represented by the codes as a relationship between any of **Model, System Breakdown, System Integration, Code**, with either of **Ideas** or **Knowledge**. Thus, I first looked for the occurrence of one of **Model, System Breakdown, System Integration,** or **and** the occurrence of one of **Knowledge** or **Ideas**, in the top 3 codes, for SWC and LWC. In SWC 5 of 25 interview participants have a code from both groups of codes, and 12 of 25 interview participants have a code from one of the two groups. In SWC only 8 of 25 interview participants have codes from neither of the two groups. In LWC 13 of 25 interview participants have a code from both groups of codes, and 8 of 25 interview participants have codes from one of the two groups. In LWC only 4 of 25 interview participants have codes from neither of the two groups. Table 5.6 shows these numbers.

When examining the top 10 codes for comparison, the relationship between one of **Model, System Breakdown, System Integration, Code and Ideas or Knowledge** was extremely apparent. In SWC 18 of 25 interview participants had both codes in their top 10 codes, and in 21 interview participants had both codes in their top 10 codes.

Second, I looked for the frequencies of alternatives and serial evaluation, to compare the two in the same manner as Section 5.1.1. When looking at the list of codes found in Appendix B, it is clear that there is no code called Serial Evaluation. Again, the interactive nature of the dictionary suggests the use of words that could be coded as Serial Evaluation did not appear to the extent that there was merit in developing a code. Equally interesting, is that when examining the presence of the code **Alternatives**, in the top 3 codes of interview participants, the code **Alternatives** only occurred in the top 3 codes for 3 out of 25 interview participants for LWC and 0 out of 25 interview participants for SWC.

The indicators from the frequencies that mental modeling and the use of consequential choice or serial evaluation merit further research, are weak, relative to similar indicators for the topic addressed in Section 5.1. In my opinion this could be a result of one of three events: First, that there is a problem with the coding or the interactive dictionary, second, that the idea just does not merit study, third, that there is a weakness in using frequencies as

a sole indicator of the important concepts. I will address the potential of the first event in Chapter 11: Validity. I will address the potential of the second event in this section by qualitatively evaluating the relationship between mental models and the use of consequential choice or serial evaluation. Last, I address the potential of the third event by re-iterating that frequencies and window size have a tendency to remove the context of a transcript when using content analysis [Kripend80].

Table 5.6: Number of Interview participants with Code in 3 Most Frequent Codes

Type of Coding	Code	# of Participants
Small Window Coding	One of Model, System Breakdown, System Integration, Code **and** One of Ideas or Knowledge	5
	One of Model, System Breakdown, System Integration, Code **or** One of Ideas or Knowledge	12
	None of Model, System Breakdown, System Integration, Code **and** None of Ideas or Knowledge	8
Large Window Coding	One of Model, System Breakdown, System Integration, Code **and** One of Ideas or Knowledge	13
	One of Model, System Breakdown, System Integration, Code **or** One of Ideas or Knowledge	8
	None of Model, System Breakdown, System Integration, Code **or** None of Ideas or Knowledge	4

5.2.2 Emerging Relationships

The Generic Relational Coding (GRC) exposed relationships between codes such as **Model**, **Code**, **System Breakdown** and **System Integration** or **Customer Product** (i.e. codes dealing with the design of an application) and codes such as **Ideas** and **Knowledge** and **Understand** (i.e codes dealing with what the designer was thinking). I present these relationships to show the prevalence of the idea of a mental model, in the interview transcripts. For readability, I have grouped the quotes below into topics that I found to be similar. At this time I consider these conceptual separations to be potential indicators of the ways in which mental models are developed, and thus are beyond the scope of this thesis.

For example, five quotes addressed separating business logic from user interface logic, when discussing mental models. There are 41 quotes in all.

The following quotes addressed the separation of logic when discussing the mental model of the system.

This mentor discussed a separation of business decisions from design decisions and that designers should focus on design decision to properly convey the design in code.
CS1.1_Q2: "… I always try to encourage people to think from other parts of the design… so one of the basics is that very isolation of the domain design decisions from other decisions."

When discussing cues that a design change needed to be made this developer discussed a design principle he had learned.
CS1.13_Q2: "The biggest thing I was seeing was that I seemed to be mixing business logic and presentation code in a common method, which isn't really the way it should be done."

This developer discussed making a decision by thinking about a higher level problem.
CS1.14_Q2: "…try to divorce myself from the actual detail UI, database, whatever, and to think on a higher level."

A desire to separate business logic from technical logic was a cue that a design change needed to occur for this developer as well.
CS1.15_Q1: "I guess the principle is, it should be really easy to build business logic, and we wanted all the computer science [logic] out."

Again a mix of presentation and business logic was unhappily commented on by this developer.
CS1.17_Q1: "The presentation was mixed with the business logic."

The following quotes addressed the idea of communication when discussing the mental model of the system.

This mentor was one of two participants who explicitly connected a person's ability to verbally communicate a piece of design with the final software design.
CS1.1_Q3: "…thinking of statements I make when I'm talking out loud about design alternatives … when I'm talking out loud my speech centre is involved, you know that piece of our brain is sort of tailor made for grammatical communication and this programming is all about grammatical arrangement of symbols to communicate certain things, so why not use that speech centre in your brain."

This mentor was the other interview participant who discussed this connection.
CS1.5_Q5: "I think it takes the human mind to say, 'alright, I think these [objects are one and the same.' … Communication, expressing ideas, is about naming things. So we look at this [code and] maybe we recognize through duplication, or maybe we just look at this and say we don't understand, 'this code just makes no sense, what's it talking about?' and we start changing the words that we used."

This mentor described communication between people about design, arguably addressing shared mental models implicitly.
CS1.5_Q6: "You use whatever it is, [diagrams, CRC cards] … so that your head and my head are in the same design space, just thinking the same ideas, and then put that in the code. Now we might need to document that for some reason. But even then, the purpose of documenting is to get it into somebody's head."

When discussing problems in code this mentor suggested that problems emerged almost naturally, when no team member challenged an idea.
CS1.9_Q2: "[Problems in code] are normally a function of one person on the team… He or she doesn't make all the mistakes but he or she set up the idiom, 'We are going to write this type of code this way.' They may not have dictated that but they said, 'Ok, I wrote this first module that does this.' And everybody looked at that as a sample and that idiom

becomes how they code that type of thing and the problem just propagates because nobody ever really looked at it and though about it in enough detail to say, 'Is this an idiom we want to put everywhere?'"

This mentor discussed a solution to a design problem with an informality that suggested he had a strong idea that this was a reasonable solution.

CS1.25_Q2: "So there's an accumulation of experience in our minds obviously from books that we had read. … [The design problem] was just sort of a collective, multithreading, throughput, messaging problem that we collectively solved together, just thinking about it."

Developing an understanding of what you're about to code is an obvious task, for this mentor.

O5.40: "You may have mastered the technology and you have good design skills. Then you have to write a program for analyzing molecular structures or whatever… and of course you don't know anything about that. In order to be effective on a software team you have to know something about that."

The following quotes addressed the idea of design patterns or principles when discussing the mental model of the system.

This mentor suggested that experience could be summed up into patterns to design solutions that could be re-used.

CS1.1_Q4: "Partly [people] are good designers because they've seen a lot of things before, so when they see a problem, they're not starting from scratch. They remember the general pattern of a solution to this kind of problem … But a lot of the value of experience is going to be hard to put down in a book…"

An explicit reference to the Gang of Four design patterns book [Gamma95] was made by this mentor, as a general reference book.

CS1.12_Q3: "I certainly have a pool, I have 10, 20, whatever it is, kinds of pattern books on my shelf, not all of which I've read yet unfortunately, but you know enough…"

Repetitious code is to be avoided as per good design principles, according to this developer.
CS1.15_Q2: I'm having the same kind of code repeated and that's the first indicator, there's something common here, we should think harder here about how to deal with the variance and put it all in one place."

Later in the interview this same developer explicitly referenced the use of a design pattern.
CS1.15_Q3: "So we came up with virtual singleton, I think we published it, which was again the idea being that you're in a thread and you need to access a singleton, singletons look like singletons to you but they're thread-based singletons."

This developer also discussed re-use of design patterns.
CS1.15_Q4: "At this point [design patterns] are just engrained and you just see things in a certain way. You're looking for common solutions and you just say 'oh, that's just like that.'"

This developer explicitly used the patterns book when implementing his design change.
CS1.23_Q2: "There's the patterns book by the Gang of Four, I certainly looked that up. Called [the pattern] domain adapters… a lot of the ideas in there came into it. More fundamentally than that, my approach was influenced by my background. I've spent some years working at Hewlett Packard. There are some very, very smart engineers working there who pushed these concepts of how to build software on us. Just put it into our heads."

This developer also said he explicitly used the patterns book.
CS1.24_Q3: "Well you get to know patterns. So basically the way I've always used the patterns book is I have a vague idea of what I want and I know which section of the design pattern book to go to, to help solve the problem."

After some thinking this developer mentioned that his design change applied some design principles.

CS1.22_Q1: "When we chose to go down the stored procedure and trigger route we did look into documented best practices … So I guess we were influenced somewhat by best practices."

A design solution to a common problem was known because of experience, according to this mentor.
CS1.12_Q4: "…so if you're going to write a compiler you pretty much know the general structure."

The following quotes addressed the idea of a random thought occurring when discussing the mental model of a system.

When asked where the idea for a design change came from this developer said the following:
CS1.10_Q2: "I think we also had to invent a lot of stuff."

This mentor suggested that designers just simply have ideas for the path a design will take.
CS1.12_Q5: "Sometimes people will have a vision for how this will work…"

The seeming randomness of some ideas was suggested by this developer who had difficulty tracing the source of his design ideas.
CS1.20_Q1: "Where did I get the idea? I don't know. There was probably an assumption on my part … I guess it's kind of a basic encapsulation idea."

The following quotes addressed an end user's ideas about the system, when discussing the mental model of a system. This developer identified a difference between what he understands about the software and what the user understands.
CS1.18_Q2: "This [end user] never understands the complexities of the software…"

This developer recognized the reaction a customer has in response to changes made in a software application.

CS1.19_Q2: "With a customer, if you give them a new release and it looks all different and they're not expecting it, they have a tendency to get a little nervous because they don't know what else has changed."

The following quotes addressed the idea of a timeline in software design, by directly or indirectly alluding to a start or end point, when discussing the mental model of the system. When asked a follow-up question to the options question, this mentor said iteration planning was a time to start learning about a problem in order to understand what is involved in the design of the feature.

CS1.1_Q5: "So during iteration planning you have all these story cards and they get estimated. Now how would you do that? The way I do that is I have to understand the feature and I have to understand enough about the design approaches we might take to get a feel for how hard it would be."

The mentor said the motivation to write a test (in Test Driven Design) comes from a new feature.

CS1.7_Q3: "There's a feature not in the system that needs to be in the system. Someone identified something for the system to do, then we get a rough idea of what objects it would take and how many objects do I think I might need."

This mentor specified that in agile design a Designer stops and thinks about the design more often than stopping and changing the design.

CS1.9_Q3: "You're stopping and thinking about [the design] more… looking at designing the next part of [the application] in smaller increments."

In describing new product development this mentor mentioned the knowledge the team had at the beginning of the project.

CS1.11_Q2: "We started out with a huge amount of knowledge of chemistry and optics and we purchased an equity position in a company … so we bought their rights to their technology of how they did bulk polymerization."

Again in new product development this mentor contributed to the idea that solutions are emergent, to accommodate a market area.
CS1.11_Q3: "You have a general market idea, you have a general pricing idea and you kind of know where you want to have a new product."

This developer explicitly referenced timeframes and knowledge in software design.
CS1.20_Q2: "I think the further out in the future you go, the more nebulous your design should be."

This developer suggested there are multiple ways to think about the same design problem and solution.
CS1.14_Q3: "The output [of thinking about a design decision] is usually to a very high level design because I want leeway for the actual developer to think this through again, maybe come up with a better solution in the same spirit."

The following quotes address the idea of rethinking, when discussing the mental model of a system.

This developer was developing a testing framework similar to JUnit, but for .NET or C#.
CS1.4_Q1: "What I wanted to do was re-think the way [the application] worked, based on how you would do things in .NET or C#".

After considering one design alternative this Designer thought about the problem again and selected a different alternative.
CS1.4_Q2: "We rethought the way the whole program worked and … we enabled a lot of other things by choosing that route rather than [an alternate] route."

The remaining quotes could not be classified into any one category, based on the data, but still acknowledge the idea of a mental model.

In a general description of design change this mentor used an analogy to highlight some problems with the way software is designed.

CS1.1_Q6: "A lot of software is written kind of like a machine, so as long as the input creates the right output … that's the only objective. But I don't think it is the only objective because there's a concept, … and if you have complex software you have to design in a way that reflects that understanding of the concept."

Again this mentor used an analogy.

CS1.1_Q7: "What I want is a design that captures some need, that reflects some understanding. You can have a program that just says put these values into these slots and pull the crank and out comes other values from another slot, or you can have a program that has some objects that reflect a model of that domain, so your application looks like a logical statement. That positions you to going forward compared to drowning."

This developer explicitly referenced ideas about good design.

CS1.3_Q4: "It was more a case of experimenting with something I thought was a better design."

This mentor suggested that Designers think about design, not the tools they use to convey design (such as code).

CS1.5_Q7: "… you're really just thinking about the design and usually whatever your abstractions are, whether they're UML diagrams or CRC cards … thinking in a very focused way about design and not about, not much at least, about the code."

This mentor almost explicitly referenced mental models in design.

CS1.5_Q8: "But the place for design isn't on the board. It's in the head."

In new product development Designers have incomplete but still sufficient models, according to this mentor.

CS1.11_Q4: "They're not exact, they're models; but they're models which a development team can understand and make decisions from."

This developer referenced a connection between understanding and technical components of a software application.

CS1.24_Q4: "We needed something [where] you could understand state better, you could understand error message better, you could process those better…"

The point of all these quotes is to show designers thinking about design and developing design solutions from past experiences, best practices, group work, and numerous other environmental factors, rather than facing a design problem and immediately knowing a solution, or immediately looking the solution up in a textbook (for example). Of all 25 interviews, only 5 interview participants did not provide a sentence or phrase that referenced the use of a mental model. From this, the above quotes and the supporting literature [Guindon90b, Adelson85, Gasson98], designers continuously shaped mental models of their design problems.

5.2.3 Responses to Questions

Given the presence of mental modeling, I now show a relationship between the strength of the mental model and the decision approach used. I intend to show that the stronger the mental model, the more a designer used serial evaluation, and the more open the mental model, the more a designer used consequential choice. I will show this through the interview participants' responses to the question on Options, with quotes from the rest of the interview where necessary. After each quote I list the problem structures that are defined and that contribute to a strong mental model, or the absence of problem structures that contribute to an open mental model, as per the definitions of strong and open mental model. The summaries for interview participants 5, 11, 19, 21 and 22 are left till the end of the discussion, to show the outliers.

Interview Participant 1: Interview participant 1 was a mentor participant and his interview contained a continual comparison between what he saw people doing when they developed software, and what he did. Interview participant 1 discussed a strong mental model by recognizing that constraints are imposed on a design as a way of selecting a

design solution without considering alternatives. He discusses an open mental model by recognizing opportunity to search for design alternatives before committing to one specific design. The supporting quote for a strong mental model imposing constraints on a design and leading to serial evaluation is,

CS1.1_Q8: "The key that I keep reminding myself and others of, is there are no limits to possible models. If you have [the design] done you could say, 'I'm setting up all the constraints because of the solution,' but you know, you better do a lot of experimenting before I'll believe that. People just settle too quickly for fear they're going to look dumb or because they just don't fully grasp how many good ideas there are."

Problem structures in Strong Mental Model: Setting up constraints because of the solution.

The supporting quote for an open mental model providing an opportunity to consider design alternatives is,

CS1.1_Q9: "…people settle on the first thing they think of. If I'm looking for some kind of solution to a design problem and I think of a good idea, … I like to have a good idea but I'm not going to stop there. I'm going to go ahead and push a little, [try] variations, other possible approaches, challenge that as being the direction you take. … A lot of the ideas that I put out there are bad ideas, probably at least half of them are just plain bad ideas but fortunately they cost the project hardly anything because they were just an idea that was put out at a meeting, draw a little picture on a board, say a sentence or two and it gets shot down, for good reason, and maybe that triggers an idea in someone else's head or in mine, or I learned why it was a bad idea."

Few Problem structures in Open Mental Model: Push a little, variations, challenge the direction you take.

Interview Participant 2: Interview participant 2 was also a mentor participant and his interview contained a continual comparison between what he saw designers doing in group decision making and what he recommended they do. Primarily, interview participant 2 believed people do not discuss their criteria for making decisions enough. He discussed a strong mental model by discussing group interactions about a decision. A strong mental

model can just be selected by a strong personality. He discussed an open mental model and the need to use consequential choice by acknowledging that members of a group don't understand the reasons behind another member's conviction for a specific design alternative. The supporting quote for both these points is,

CS1.2_Q5: "…often times its who screams the loudest and they win, and what happens is you go through the same screaming exercise time after time after time and the reason you do that is you never really stop to try to understand why the other person thinks the way they do – what is their criteria for making a decision. The more you can get down into the little, why is it that I think this is important and why is it you think this is important… what's our criteria for this, why are we doing it this way, what is it we're really trying to understand here…"

Problem structures in Strong Mental Model: screams the loudest and they win.

Few Problem structures in Open Mental Model: try to understand why the other person thinks the way they do.

CS1.2_Q6: "The traditional way of designing and going through projects has usually been, 'well, let's figure out the designs upfront, make those decisions, and then we'll live with them… So you may consider some options upfront but once you've decided on something then you go forward with it. The pressure has been to make early decisions. The agile, lean way of doing things often times is to keep multiple options open as late as you can into the project. … You don't want to make [the decision] until you have to, because you can always accumulate more information up to that point."

Problem structures in Strong Mental Model: Once you've decided something you go forward with it.

Few Problem structures in Open Mental Model: You can always accumulate more information.

Interview Participant 3: Interview participant 3 spoke from a developer perspective about a design change to refactor a Java application, that dynamically generated HTML code, to an application using XML and XSLT. Interview participant 3 knew about XSLT in general, bought a book on the topic and learned about it, sold the idea to business managers, and the team made the change. Interview participant 3 had a strong mental model that involved

XSLT and supported existing constraints. In addition, he used primarily serial evaluation. Supporting quotes from the Decision question and the Options question are,

CS1.3_Q5: "I'd use XML and I think I'd played around with XSLT a little bit but never on a production system. I went out and bought a book, the first book on XSLT … and I just started devouring that book.", "The company being in SAP, they needed to have a customizable look and feel for their website. So that alone made me think that XSLT would be a better approach… You write off a lot of other options because of their coupling to different vendors… The reason that I liked the XML, XSLT approach was that I could see right away that it was very independent, open source so there were no fees associated with it, it didn't couple us to any vendor…so I thought a little bit about other design approaches, but mostly decided pretty quickly that it was a good way to go."

Problem structures in Strong Mental Model: Bought book, SAP, no coupling to vendors.

Interview Participant 4: Interview participant 4 discussed changing the design of NUnit (a unit testing framework for .NET and C#) to use an aspect of .NET called custom attributes. Interview participant 4 spoke from a developer perspective. In this design change one other alternative (besides custom attributes) was discussed but only the custom attributes approach was prototyped. Interview participant 4 had a well defined mental model due to numerous constraints he imposed/selected for the design solution and the past experience of numerous people. Again, only one other alternative was considered and not significantly. Supporting quotes taken from the Decision question and the Option question are,

CS1.4_Q3: "[The original NUnit] didn't really take advantage of any of the idiom that the .NET backbone itself provided, and what I wanted to do was rethink the way it worked based on how you would do things in .NET or C# … Recognizing the capabilities [of .NET] was really the big control of the process. … A lot of the people that I had worked with and was working with at the time had all used JUnit on Projects in Java and wanted a similar tool. … We looked at completely separating the meta information to a separate file, so it'd be some sort of XML file. The problem there was that we ask developers to write the tests and if we made two separate files we would have to write something that would bring the files together… We didn't [consider anything else] because we wanted to maintain

something similar to the JUnit experience. … We talked about the XML approach, but then rejected it."

Problem structures in Strong Mental Model: Similar to JUnit, capabilities of .NET.

Interview Participant 6: Interview participant 6 spoke from a mentor perspective and discussed practices of agile methods throughout the interview. He identified the difference between open and strong mental models when discussing iteration planning meetings and the learning that occurs over the course of an iteration. Options are weighed primarily during the iteration planning meeting and, if necessary, changed during the iteration. Given that the iteration planning meeting by definition involves discussing new features, and given that the course of an iteration involves implementing these new features, it follows that the mental model of a system that a designer has or a group of designers have, strengthens over the course of an iteration.

CS1.6_Q5: "I think where we see those trade-offs most formally done is in the iteration planning meeting. Because that's when they're taking a story and breaking it down into tasks. Its very common that we will see some UML drawn to make sure that people have a common view of what we're talking about. So in those discussions, trade-offs and options are weighted. [Sometimes] they don't spend enough time on something, but it's rare that that mistake would ever live beyond the iteration because as soon as they get into it they'll see how, 'this was a bad idea and we should have talked about this more.'"

Few Problem structures in Open Mental Model: We should have talked about this more.

Interview Participant 7: Interview participant 7 spoke from a developer perspective and discussed continuous use of simple rules such as remove duplication and always using test driven design. His support of strong mental models leading to serial evaluation and open mental models leading to consequential choice was explicit.

CS1.7_Q4: "Often, if I'm quite confident that I know a good solution that does what I need, I will just do that, and that's a question of experience. If I've seen it before and I know where its going then I just go ahead and get there. A lot of the time though, all I have are these rules and I don't necessarily know in advance, if I apply this rule, where am I going to end up? I know that I can do four things here, I don't know where any of those four

things will lead me. So let's just see what happens. So it's a lot of very small experiments with the goal of accumulating the experience and heuristics that I can use next time to then inform my decision."

Problem structures in Strong Mental Model: Experience leading to confidence.

Few Problem structures in Open Mental Model: "Let's see what happens."

Interview Participant 8: Interview participant 8 spoke from a developer perspective and discussed part of an application where new types of fees were added periodically to the application, when the customer requested. Adding fees required modifying source code, which was, "a substantial amount of work" that involved changing numerous interfaces. The design change implemented was to use XML to describe a fee and to use XSLT to transform XML to a calculation that outputs the type of fee. Interview participant 8 showed a strong mental model by requiring a language that Java could run for the design change. Like interview participant 3, interview participant 8 though a bit about alternatives but decided quickly on the selected alternative. During the interview he also never explicitly identified another alternative that was considered, despite initially saying he 'bounced ideas' around with another developer, and despite being asked the question twice. Supporting quotes are below.

CS1.8_Q2: "[One other developer and I] were just sort of bouncing ideas off of each other talking about … what could be used and then went on the internet and started searching for, … basically we were looking for some language that Java could run, which XSLT is. And that was the one that popped up first and I started doing a proof of concept and it worked out pretty well and I just sort of ran with it. … It was pretty close to the first [alternative] that popped up on our radar when we started looking. And it ended up working and it was the first one I tried so I didn't really look at too many other options, no."

Problem structures in Strong Mental Model: A language Java could run, prototype worked well.

Interview Participant 9: Interview participant 9 spoke from a mentor perspective and discussed primarily what he saw designers doing, and his opinions of what he saw. When

discussing a strong or open mental model, interview participant 9 discussed the size of a design problem – being a big problem or a small problem. This relates to the idea of ill structured problems or well structured problems in that small problems are arguably more well structured. In this respect this interview participant supports the discussed theory at least partially. The connection to understanding (i.e. mental model) a big or small problem is not explicit.

CS1.9_Q4: "How often do I see [designers considering alternatives]? I would say most people do that pretty well on the big things. … I see people trying to do that right at the early complex level, you know, where it's vital and important to a situation. This is something I'm going to live with forever. If it's a very isolated part I don't see people thinking about it that much but I don't know that that's a problem. I mean if it's a very isolated part and I just use a linked list, what the hell, so what, I can replace it later, and it may not be worth a whole lot of thought."

Few Problem structures in Open Mental Model: The early complex level.

Problem structures in Strong Mental Model: Isolated part of the system.

Interview Participant 10: Interview participant 10 discussed three different decision problems, all from a developer perspective, and all concerning the same business problem. A customer was using a system implemented by interview participant 10's company in tandem with another system implemented by a third party company. Each time the customer had to enter data, s/he had to enter it twice. The three design decisions interview participant 10 worked on were a)the points of workflow from a user interface perspective, that data would be traded from and how it would be traded b)the supporting class hierarchy or design model and c) the technology to transfer the data (e.g. COM, web services). In the first situation interview participant 10 used consequential choice. In the other two situations interview participant 10 used serial evaluation. Supporting quotes are below.

CS1.10_Q3: "After analyzing [the first situation] for a while I decided there were two main situations where we had to integration with [the third party system]. … We went pretty far down the line of developing some options as quick prototypes, just to see how easy it was going to be or how it was going to work out." Here the interview participant discussed three alternatives and prototyping.

Few Problem structures in Open Mental Model: After analyzing, see how it was going to work out.

CS1.10_Q4: "[In the second situation] some of the underlying … design was taken out of patterns books, design patterns. The subscriber and events thing… is one example. We noticed that other programs, like this external third party one, would want to be notified when events happened in our program. So you just look at these design patterns and one of them is a subscriber model… I think once we found that design pattern and decided it fit really well, ya, I don't think we played with that [decision] very much."

Premise in Strong Mental Model: Fit really well.

CS1.10_Q5: "[In the third situation] there were also all kinds of technologies that allow you to pipe the data in between, COM, web services or whatever, all kinds of crazy stuff. … The decision to go with COM… we were heavily leaning towards that area in the first place because of the programming language we were already using in our main product and although we had a choice of quite a few different technologies … we went with that because it was by far the easiest for that programming language to use."

Premise in Strong Mental Model: Heavily leaning towards the idea, programming language already used, easiest.

Interview Participant 12: Interview participant 12 spoke from a mentor perspective and discussed what he saw designers doing and what he has done in different design scenarios. The idea of a strong mental model is explicit through reference to implementing compliers and how to implement compliers being "a done deal". In this situation, options may be tried across a career, but not in a single implementation of a compiler, so serial evaluation is actually employed. The idea of an open mental model is shown explicitly as well and leads to the use of consequential choice. The supporting quote for both points is as follows:

CS1.12_Q6: "Sometimes [I consider alternatives], other times its just, like for me, compilers, mini compilers is something you end up writing over and over and over, different times. You can say I'll try Lex this time, or I'll try regular expression this time, you know, over your career you have different opportunities to try them and you probably hone in on other ones you're familiar with. And sometimes during a project you'll have times where you'll say, 'I've looked at these 3 things. I have no idea which is better you

know, let's just do an experiment and find out where it goes. … Sometimes it's just lay them out on paper, sometimes it's go try coding it up 3 different ways for a while…"

Problem structures in Strong Mental Model: The design of compilers is a done deal.

Few Problem structures in Open Mental Model: Having no idea about a design solution and trying out alternatives.

Interview Participant 13: Interview participant 13 spoke from a developer perspective and discussed a project he was working on in his personal time (i.e. not related to his software job). He encountered a design problem where the business logic was mixed with the presentation logic. Interview participant 10 had a strong idea that these two kinds of logic should not be mixed so he refactored the code. He did not consider alternatives. The supporting quote is below, taken from the Decision question and the Options question.

CS1.13_Q3: "The biggest thing I was seeing was that I seemed to be mixing business logic and presentation code in a common method which isn't really the way it should be done. The user interface should be separate and the business logic should be separate. … I wasn't using the best object-oriented principles. … No I didn't really consider anything else. I could have left [the code] all in one page. But other than that I didn't' really consider anything else."

Problem structures in Strong Mental Model: Object-oriented principles and "the way it should be done."

Interview Participant 14: Interview participant 14 discussed a design change from a developer perspective. There was an existing feature in a legacy system that wasn't easy to use and not familiar to users (because it wasn't easy to use). Interview participant 14 decided to completely replace the code. While he knew of other alternatives (e.g. work around the problem, retro-fit tests) he considered the decision an easy one and did not actually consider alternatives.

CS1.14_Q4: "For that particular application there was no other easy decision. … It wasn't an option decision. Usually I spend several hours on major design decisions … but that one wasn't hard to make."

Problem structures in Strong Mental Model: Easy decision to make.

Interview Participant 15: Interview participant 15 discussed a design change to move all caching into one general purpose cache, because logic was getting spread throughout code and it was taking a long time to run unit tests. He spoke from a developer perspective. Interview participant 15 spoke *in hindsight* about alternatives but at the time of the design change did not consider alternatives because the selected alternative was "reasonable". Interview participant 15 was also under time pressure.

CS1.15_Q5: "We could have tried to get more resources. … We could have tried to add more hardware… we could have used a mock object approach. At the time we weren't really aware of the mock object approach. We didn't use mocks, mostly because we didn't know about them. We were used to stubbing things out. We didn't have any other reasonable alternative."

Problem structures in Strong Mental Model: No other reasonable alternative.

Interview Participant 16: Interview participant 16 spoke from a developer perspective about ensuring method signatures matched between interfaces he created and code provided by a third party organization. As a programmer, his job was extremely focused on small code tasks, and thus the mental model was strong. In general, interview participant 16 did not talk about this design experience in a positive fashion citing immense time pressure. When asked if he had any alternatives in the decisions he made he said sarcastically, and somewhat cynically,

CS.16_Q1: "Ya, introducing 1 method or 15 instead, so 1 method with 50 parameters or 15 with 3 parameters each. Ya, there were options. There was still somebody who was in charge, the project leader, who would have to approve everything. In some cases it would be easy, in some other cases it required 5-6 emails."

Problem structures in Strong Mental Model: Method definition to match an interface, project leader approval.

Interview Participant 17: Interview participant 17 discussed design from a developer perspective and discussed a design change that involved moving from one design paradigm to another. While interview participant 17 recognized that it was a big design change, he

also discussed knowing what the problem with the old design paradigm was, and knew of a solution from past experience. Given this, the mental model was somewhat strong. interview participant 17 considered two options. The supporting quote is below.

CS1.17_Q2: "…the presentation [code] was mixed with the business logic. … We were faced with either refactoring to use taglibs or jsp, or going in the velocity [template] way. … It was about planning for the future and also about trying new stuff. … I can't remember all the reasons we chose the velocity way over the taglibs way. Basically there was a paper that was about that – like why use [velocity templates] and that's what we based it on. There were probably other choices we could have made. … We had to do a proof of concept first. Like, we did, 'here's the before, here's the after' and we got it working with one page in the old system so once we had it working then I think it was an easy sell."

Few Problem structures in Slightly Open Mental Model: Easy sell when it was working, desire to try something new.

Interview Participant 18: Interview participant 18 discussed design changes from a developer perspective. He explicitly identified the strength of a mental model leading to serial evaluation by identifying constraints imposed on some of the design decisions he had to make. Similarly he identified the use of consequential choice in areas where the mental model is not as strong.

CS1.18_Q3: "Somethings are not an option. For example, because we're a Microsoft house and we've got people in the office that are certified on sequel server there was no real option between Oracle and DB2 and sequel server. … Now other things, like say credit card processing, we have to weigh. We just had [a developer] do a case study a month ago for what the new options are for credit card processing. He went through all the different providers and he did a cost analysis, how much is it going to cost us to implement, how much is it going to cost people to use, how new is the product. … So if its backend stuff, depending on what it is, we'll actually take 3,4 days, sometimes a week, and prepare a study and … we'll all read it and go over it asked on what that one guy's research was."

Problem structures in Strong Mental Model: Microsoft house means Microsoft product.

Few Problem structures in Open Mental Model: Did not know about credit card providers

Interview Participant 20: Interview participant 20 discussed design changes from a developer perspective, and discussed building an aspect of configuration into an existing application. He used consequential choice and the primary way in which an open mental model was shown was through an open group mental model. Interview participant 20 had an idea for the design change but had to defend it to another developer who had another idea. The supporting quote is below.

CS1.20_Q3: "Ya, I did actually [consider alternatives], one of the other team members was arguing heavily to leave one of the classes the way it was but that was in an effort to make the testing easier. … For the real solution it was an unreal scenario. … So it's kinda a false sense of security. … We definitely would have gone down the wrong road."

Problem structures in Strong Mental Model: Definitely gone down the wrong road.

Interview Participant 23: Interview participant 23 discussed a design change from a developer perspective. The design change began with a feature request for interview participant 23 called a "one-off". It was a feature that was important only on one small domain, and he did not see a lot of value in it. So interview Participant 23 actually made the feature request more challenging by trying to solve it in a way that could be reused. In this sense he had a looser mental model of the system than he otherwise would have had. Consequential choice was used. Supporting quotes are below, from the Decision question and the Options question.

CS1.23_Q3: "I'm always looking for solutions and designs that will allow us to accomplish multiple goals with the same project because we can't afford to do one-offs. … So when this particular project came along to come up with a, what I considered to be, it wasn't worth doing. But I was overruled. So I set out to find a way to solve the problem in a way that would allow us to not waste our time doing it. To get some other goals out of it. … Some initial conceptions of [the design] were simply to produce a path and then … take that path or interpret it, get the data or put the data and it basically was a thing that a treewalker would go through. So that was probably the first idea that was tossed out. And it probably would have been do-able probably quite a bit less effort to hook into. But it would've been a solution to a particular problem. It wouldn't have been reusable. … I

know that other people started playing around with some ideas that got abandoned when I took charge of this."

Few Problem structures in Open Mental Model: Get some other goals out of it.

Interview Participant 24: Interview participant 24 discussed a design change from a developer perspective. He was required to improve an import structure in an existing application to handle a large influx of data. Time constraints were tight. It was a problem that he always knew about; he had been reading books on the technology and design patterns and used some past experiences to help him. Interview participant 24 thus had a strong mental model, and explicitly discusses using serial evaluation because he just knew what he was doing was right. Supporting quotes are below.

CS1.24_Q5: "I think there comes a point where you know something is going to be the right thing for the right task, the right solution for the task at hand. The other option is stick with what we did prior, that's always an option. Is there a better solution to this problem? I never thought about it. … It was the best solution to the problem that we had, with the time given to us to make the decision. Give me another six months, maybe I'll come up with a better design than that. Ya, but this was not one of those issues that you have a lot of choices for."

Problem structures in Strong Mental Model: The best solution to the problem with the time given.

Interview Participant 25: Interview participant 25 discussed two design changes, both from the developer perspective. One design change was a personal project, the other was a work project. Interview participant 25 explicitly identifies the use of serial evaluation when there is a strong mental model and consequential choice when you have a more open mental model. The supporting quote is below.

CS1.25_Q3: "…when you've got a large production system you are a lot more constrained sometimes with the options that you can use. In this fun pet project where I was very interested in exercising one aspect of the design I really set myself up from the beginning so that I would have a lot of options cause I wanted to try different things and really kick the tires and be able to speak intelligently. A production system, up there running for a long

time, you're not as, you can't be, it's not as free-wheeling obviously because you've got tons of thousands of more lines of code … the architecture's much more entrenched."

Problem structures in Strong Mental Model: Production System means more constraints.

Few Problem structures in Open Mental Model: Set up from the beginning to try different things.

Outliers: There were two case studies that did not fit the theory that a strong mental model leads to increased use of serial evaluation and an open mental model leads to increased use of consequential choice. These were interview participants 5 and 22. Interview participant 5 discussed the idea of satisficing, and suggested this would occur regardless of the state of the mental model. Interview participant 22 discussed consequential choice, despite having a strong mental model of the design change. Quotes are below for each interview participant.

CS1.5_Q9: "You should have a very full bag of tricks, you should know as many things as you can, but you shouldn't reach very deep into that bag for solutions. You should reach only as far deep into that bag as you must. … I think if a person is not very experienced in whatever they are working in … I think they are well off to pick a simple solution because its in building the bridge between one side of the river and the other that we learn most about getting across the river."

Interestingly , interview participant 5 also said,

CS1.5_Q10: "In a sense, I think everyone should do this: whenever I start thinking about a problem, between the first moment the problem gets into my head and the moment my fingers hit the keyboard, I'm thinking about alternatives."

CS1.22_Q2: "Ya, we did look at better ways of coding it locally, we looked at trying to cache some information, loading some information beforehand, things like that. … We probably spent say a week in total looking at solutions and identifying the strengths and weaknesses of them before we settled on everyone agreeing that it would be best to put it into the database."

One point noted here is the phrase “everyone agreeing”. I leave interview participant 22 as an outlier at this time, but refer to this interview again when I discuss lessons learned from the second empirical study.

Table 5.6 shows all interview participants, the problem structures they used in discussing design change, and their respective references to serial evaluation or consequential choice. Using the definitions of strong and open mental models, the relationship between a strong mental model and serial evaluation, and the relationship between an open mental model and consequential choice, are clear. Some of the participants referenced both relationships and both references are shown in Table 5.6. The outliers are highlighted in grey. Interview participants #5 and #22 have been discussed, but also highlighted are interview participants #11, #19, and #22. These outliers are discussed in Section 5.3.

5.2.4 Rational Decision Making, Naturalistic Decision Making

The interviews were analyzed with a lens on rational decision making and naturalistic decision making. I searched the interview transcripts for one of two points to support the idea of the strength of a mental model impacting the approach to decision making. I searched for indicators of consequential choice or serial evaluation (Table. 4.1 points 1.2 and 2.2). As per the initial interpretations in Table 4.1, serial evaluation is an indicator of naturalistic decision making and consequential choice is an indicator of rational decision making. As shown in Table 5.7, there was no consistent alignment with rational or natural decision making across the interview participants. This directed me towards searching for factors affecting the approach to decision making, such as the structure of the design problem [Zannier07b] or the mental model the designer maintained.

5.2.5 Summary of Second Conclusion via Interviews

The first empirical study used interviews as evidence that the designers’ approach to decision making depends on the strength of their mental model of the design problem. This section (Section 5.2) has contributed to this point in four ways.

Table 5.6 Summary of Decision and Approach

ID	Problem structures in Decision	Approach to Decision
CS1.1	Setting up constraints because of the solution.	Serial Evaluation
	Push a little, challenge the direction you take.	Consequential choice
CS1.2	Screams the loudest and they win.	Serial Evaluation
	Try to understand why the other person thinks the way they do, can always accumulate information	Consequential Choice
CS1.3	Had idea, bought book, prototyped, sold to managers.	Serial Evaluation
CS1.4	Built new test framework based on JUnit and capabilities of .NET and C#.	Minimal Consequential Choice
CS1.5	Make atomic changes that make design better globally.	Consequential Choice & Serial Evaluation
CS1.6	We should have talked about this more.	Consequential Choice
CS1.7	Experience leading to confidence.	Serial Evaluation
	Let's see what happens.	Consequential Choice
CS1.8	A language Java could run.	Serial Evaluation
CS1.9	The early complex level.	Consequential Choice
	Isolated part of the system	Serial Evaluation
CS1.10	After analyzing, see how it was going to work out.	Consequential Choice
	Design pattern fit really well	Serial Evaluation
CS1.11	New product development	Serial Evaluation
CS1.12	The design of compilers is a done deal.	Serial Evaluation
	Have no idea about a design solution, try alternatives.	Consequential Choice
CS1.13	Best practices, "the way it should be done."	Serial Evaluation
CS1.14	Easy decision to make.	Serial Evaluation
CS1.15	No other reasonable alternative.	Serial Evaluation
CS1.16	Method definition to match an interface, project leader approval.	Serial Evaluation.
CS1.17	Easy sell when working, desire to try something new.	Minimal Consequential Choice
CS1.18	Microsoft house means Microsoft product.	Serial Evaluation
	Did not know about credit card providers.	Consequential Choice
CS1.19	Test improvements made to poor workflow.	Serial Evaluation
CS1.20	Definitely gone down the wrong road.	Serial Evaluation
CS1.21	Refactor existing user interface under time pressure.	Serial Evaluation
CS1.22	Moved logic from a middle tier of the architecture to a stored procedure.	Consequential Choice in group.
CS1.23	One-off project developed so it is useful elsewhere.	Consequential Choice
CS1.24	The best solution to the problem with the time given.	Serial Evaluation
CS1.25	Production system means more constraints.	Serial Evaluation
	Set up from the beginning to try different things.	Consequential Choice.

Table 5.7: Classification of Response to Options Question

Interview ID		1*	2*	3	4	5*	6*	7*	8	9*	10	11*	12*	13	14	15	16	17	18	19	20	21	22	23	24	25
Options	RDM	X	X				X	X		X	X		X		X		X	X	X		X		X	X		X
	NDM	X	X	X	X		X	X	X	X	X		X	X	X	X			X						X	X

- Through frequencies of codes discussing design (e.g. Model, System Breakdown) and codes discussing understanding (e.g. Ideas, knowledge) being somewhat prevalent in the top three codes of each interview participant.
- Through 41 quotes qualitatively showing the relationship between these same types of codes (e.g. Model – Knowledge)
- Through 20 case studies showing the strength of the mental model impacting the approach to design decisions
- Through a mix of the use of traits of rational decision making and naturalistic decision making indicating other factors required examination.

5.3 Study-Specific Results

I build impressions of aspects of decision making from the interviews, which at the time could not be captured in a general conclusion. These were observations I brought to the second and third studies and looked for affirmation or contradiction. There are four observations from this first empirical study. The first is that while time pressure in making decisions was not as prevalent as one might initially think, time pressure imposed the use of serial evaluation on making design decisions. The second observation was that external goals imposed the use of serial evaluation on design decision making. The third observation was that there was disagreement among the interview participants about accessing knowledge and using knowledge. The fourth observation was that rational decision making and naturalistic decision making were both inaccurate descriptions of software design decision making for the data that this study provided. I will support each of these with quotes from the data.

5.3.1 Time Pressure

Below are the case study summaries and supporting quotes of interview participants who discussed time pressure and its role in decision making. In Section 5.2 case studies of each interview showed that the strength of the designer's mental model impacted his/her use of consequential choice or serial evaluation. For three interview participants (#11, #19, #21), time pressure was a larger factor than the strength of the mental model held. All three of these interview participants used serial evaluation.

Interview Participant 11: Interview participant 11 discussed design change from a mentor perspective. She discussed new product development from the conception of a product idea to the initial release of the product. When developing a new product the market is assessed for what it needs and a product is released as quickly as possible to generate feedback from the target audience.

CS1.11_Q5: "We didn't come out at the beginning and say, 'This is what we're going to do.' We came out at the beginning and said, we think there's some market here, we're going to figure out how to hit it, we're going to do some parts of the market. … How many options you consider is a function of how fast you got to get to market. … typically… when you bring a product to market, you don't do a lot of options. You want to have something in the market that proves the concept."

No Problem structures in Open Mental Model: Think there is a market here.

Interview Participant 19: Interview participant 19 discussed design changes from a developer perspective, who was working on testing. She discussed a large legacy system that was being improved in aspects of its workflow and formatting consistency. Interview participant 19 tested design changes made by a developer, under time pressure. Given this time pressure she asked that the team not try different options.

CS1.19_Q3: "From a testing perspective we said, 'please, no.' … If we would have started doing this at the beginning of the release cycle we probably would have given [the developers] a lot more leeway [to try options] because we would have had more opportunity to see it. … We [had to be] ready to release in 3 weeks. So that only gave us a

little time to play with [the system] and work with it. So that was a limiting factor, a constraint."

Problem structures in Strong Mental Model: End of release cycle.

Interview Participant 21: Interview participant 21 spoke from a developer perspective about making a user interface more generic and easier to edit when requests from the customer arrived. His ideas came from his imagination and from some past experience. Interview participant 21 did not consider any alternatives and said time pressure was a factor.

CS1.21_Q2: "Ya, most decisions are under the gun. Time was a factor, a big factor. I had a week."

Problem structures in Strong Mental Model: Past experience, top of his head.

When examining time pressure and its role in design decisions the majority of the interview participants did not consider time to have been a serious factor in making their decisions. These three participants were the only three that acknowledged time pressure was a constraint. Other participants (i.e. interview participant #2, #5, #15, #16, #24) acknowledged that time pressure existed but did not suggest that it impacted the decision. For example, interview participant 2 said,

CS1.2_Q7: "Everything always has some time pressure. And what I think people have to do is, respond to those time pressures but not let the time pressure force them into bad decisions."

Interview participant 15 recognized time pressure but not to the extent that he thought it impacted the work.

CS1.15_Q6: "There was a lot of pressure. I didn't feel the pressure 'cause we were making a lot of progress. … There was a lot of activity, people were working pretty fast, things were going and that was kind of scary, that you could watch the system go that fast with that many people and it would actually work."

Given these results, the first result from the interviews is that while time pressure was not a significant factor in making design decisions, it did impose the use of serial evaluation, when designers felt the time pressure.

5.3.2 External Goals

Similar to time pressure, external goals forced a decision maker towards the use of serial evaluation. External goals were any outside pressures a designer had to consider when making a decision. The majority of the interview participants (16 of 25) discussed some external goal impacting their decision [Agile 06]. What type of external goal impacted the decision varied across the interviews. Below are quotes from three interview participants who discussed serial evaluation in light of external goals.

CS1.3_Q6: "I pretty much sold it to the managers, the idea of moving to this [design] approach. And I explained various business reasons why it would be important to do so."

CS1.9_Q5: "…the decision rarely gets made in the best way. … These political or other considerations factor in and sometimes they're software related and sometimes they're not. … it's hard to even know what's right when you have all that other stuff going on."

CS1.10_Q6: "I wouldn't' say they were even technical reasons, [choosing an alternative] was because of political pressures. These other guys we were integrating with wanted to use XML because they had this new and upcoming interface to their product that imported and exported XML. And it was brand new and no one was using it yet and they wanted to get it out there and get good press. So we had pressure from them to use XML. So that was the definitive reason we went that route."

5.3.3 Knowledge

The feedback concerning how designers accessed knowledge to make a design change was extremely mixed. Some developers reported having a general awareness of ideas, or the absence of an actual search for knowledge. However, some mentors reported searching for knowledge or actively seeking it out. There was not a clear disagreement between the mentor and developer perspectives but in general the mentors spoke in a more positive

fashion about actively searching out knowledge, than the developers did. Example quotes are below, distinguishing developer and mentor perspectives.

"I don't think people come to a design problem and then look it up in … [a] book. At least not when they're expected to." Mentor Perspective

"I'm appalled sometimes at how little people read, how little people go after knowledge .. in organizations." Mentor Perspective

"I'm always amazed at the teams I work with that won't read. It drives me nuts. … I don't see any commonality in what people look at in books. I don't see them looking anyway – which drives me nuts! It really does." Mentor Perspective

"I never use those books. … You talk with purists and they say … every programmer has read 30, 50, 100 books about this and every year there's new bibles coming out. … I grew up in software, I guess it comes to me naturally." Developer Perspective.

"I haven't read a lot of books but there are a few I have read. … You can read but until you actually put it into practice you never know, sort of wonder, if what you read is right or if you even remember it." Developer Perspective

"I bought the book to implement the design idea." Developer Perspective.

These points are evidence for the third result: there is much disagreement among the interview participants about accessing knowledge and using knowledge. I found little pattern about whether or not designers actively search for knowledge, until I saw a relationship emerging between **Knowledge** and **Satisfice**. Not every interview participant showed this relationship, but five interview participants did (mentors and developers) suggesting that there is a subjective threshold, where a designer has learned enough to get by. Example quotes are below.

"You're the software expert, but you have to learn enough to have an intelligent conversation with the domain expert."

"You should have a very full bag of tricks, you should know as many things as you can, but you shouldn't reach very deep into that bag for solutions. You should reach only as far deep into that bag as you must."

"…it's a combination of mocking stuff up, writing code, web searches, there's a trigger that says, 'do you understand the story enough to do it and if not, what else do we need?'"

"…you need to know enough to make sure that the technical people on the team know what they're doing."

"I had learned enough to get into trouble."

5.3.4 Rational Decision Making, Naturalistic Decision Making

The fourth result was that none of rational decision making, naturalistic decision making, or some combination of the two is an accurate description of software design decision making. Indicators of satisficing and serial evaluation are sufficient to conclude that rational decision making is not an accurate description of software design decision making. Indicators of optimizing and consequential choice are sufficient to conclude that naturalistic decision making is not an accurate description of software design decision making. The potential to combine the two approaches works for two components of each approach: optimizing versus satisficing and consequential choice versus serial evaluation (Table 4.2, points 1.1, 2.1, 1.2, 2,2). However, when examining how time pressure impacts design decisions and how knowledge is managed in design decisions (Table 4.1 points 1.3, 2.3-2.6, 1.4, 2.7-2.8) there is much variation in the results and a specific combination of the two approaches cannot be found in our data. Thus, the fourth result is that none of rational decision making, naturalistic decision making or some combination of the two is an accurate description of software design decision making.

5.4 Chapter Summary

The results of the first empirical study are evidence for the two main conclusions and a set of eight smaller results. The first conclusion is that software designers appropriate design solutions approximately as often as they strive for an optimal or a boundedly optimal design solution. The second conclusion is that the approach to a design decision depends on the strength of the designer's mental model. The stronger the mental model, the more serial evaluation was used. The more open the mental model, the more consequential choice was used. These two points were shown via frequencies of codes, emerging relationships, answers to interview questions and aligning results with components of rational and naturalistic decision making. The first two results were that time pressure and external goals impose the use of serial evaluation on design decision making. The third result was that there is disagreement in the software engineering community about accessing knowledge and using knowledge. The fourth result was that none of rational decision making, naturalistic decision making, or some combination of the two is an accurate description of software design decision making. Given these results from the first empirical study I present the second study.

CHAPTER SIX: PREPARATION FOR AN EMPIRICAL STUDY of DESIGN DECISION MAKING via ON-LOOKER OBSERVATIONS :

The second empirical study consisted of conducting observations and analyzing field notes of the observations via content analysis and explanatory case studies. This multi-case study consisted of observations at three software organizations. The observations were the type of an on-looker [Patton02] with periodic question and answer periods to understand the varying situations observed. They lasted 9-10 days at each organization. Content analysis was used to code paragraphs, and the dictionary was different than that used in the first empirical study [Kripend80]. However, like the first empirical study, the results were compiled to build an explanation about design decision making, as it relates to real options theory [Lasher06, Yin02]. This chapter describes data collection via descriptions of the approach to the observations, and a description of the companies where the observations occurred. This chapter then describes data analysis via a description of the approaches to analysis. Last, this chapter describes my experiences with qualitative inquiry after this second empirical study. In this study, a “case” or “unit of analysis” is one observed occurrence of a design change or potential design change within an observed small software team.

6.1 Method of Data Collection

The observations were conducted from January to April 2006. In order to find companies willing to be observed while they worked, I used convenience sampling [Patton02, p 241] observing whichever companies would allow me to observe them. At one of the organizations I had a personal contact, at the other two organizations I advertised on a professional newsgroup and received two responses. While convenience sampling is potentially the least desirable sampling technique [Patton02, p242], I considered myself fortunate that it was successful (i.e. that these organizations agreed to participate). Prior to convenience sampling I attempted a form of random sampling: I contacted software organizations found in the phonebook, via telephone or email, alphabetically. This was entirely unsuccessful. Thus, I was pleased to have participants, regardless of the sampling technique.

I used only field notes as my approach to data collection. I did not use software artefacts. The main reason for this is the difficulty of finding willing companies. Again, I was pleased to have participants and did not wish to risk their willingness. At the second company I was required to sign and non-disclosure agreement and at the third company I faced some initial challenges with attaining final approval for my presence in the organization. These two experiences only strengthened my hesitation to ask for access to software artefacts. While software artefacts would have made each case study even richer, their absence does not nullify the value of the field notes that I present in this thesis.

6.1.1 Description of Software Organizations Observed

Throughout the discussions of the three organizations I refer to them as Company A, Company B, and Company C. I provide a summary of the basic statistics of each organization in Table 6.1.

Company A was a small software organization developing point of sale systems for the restaurant industry. There were three developers, one of whom was also a manager, and an additional in-house sales, management and support staff of six people. The development tasks were broken down and allocated to individuals: one developer focused on back-end tasks, another developer focused on user interface tasks. The developer who was also the manager provided support on the back-end of the system where needed (he had developed much of the system before the company grew) and helped develop the user interfaces of the system. Company A was not familiar with agile methods [Agile07] but due to the size of the organization they exhibited some traits of agile methods such as communication over heavy documentation and a workspace that was open, (i.e. no cubicles) facilitating continuous communication [Agile07].

Company B was a small software organization developing logistics software for the supply-chain industry. There were six developers, one of whom was also a manager, and an additional in-house sales, management and support staff of two people. The development tasks were broken down and allocated to the entire development team, as a whole. Company B followed agile methods – specifically Extreme Programming [Beck01] - in a

regimented fashion. Pair programming of tasks was performed the majority of the time with five of the six developers sitting at two tables together, so that continuous communication could (and always did) occur. There was no pre-established distinction of who worked on what aspect of the system. Instead, a developer would take a task from the list of tasks and ask someone to work with him. Alternatively, a developer with no task under development would pair program with someone else developing alone.

Table 6.2: An Overview of Observed Software Companies

Parameter	Company A	Company B	Company C
Breakdown of People Observed	2 developers 1 Manager/developer	5 developers 1 Manager/developer 1 Customer Contact	3 developers 1 developer/Lead developer
Number of People Observed	3	7	4
Total Number of Designers at Organization	3	6	11-14
Days Spent with Each Participant	3	1-2	2
Days Spent at Each Organization	10	9	9
Role of Agile Methods in Organization	No Role	Heavy Role	Light Role
Division of Labour	Individual	Group-based	Individual

Company C was a slightly larger organization than Company A and Company B and developed integration products for information technology operations management. There were a minimum of 11 and a maximum of 14 developers (hiring occurred during the observations) and an additional in-house sales, management and support staff of over 10 people. The development tasks were broken down and allocated to individuals. Each developer was assigned an integration pack on which to work and was matched with someone who had previous experience in the area that the integration pack concerned. This secondary developer could take over the development task should the primary developer no longer be available. The secondary developer was called a shadow. The use of a shadow was a new practice for the development team, beginning the day I began observations. The shadow had his/her own integration pack s/he was responsible for, attended meetings

between the primary developer and project managers concerning the primary developer's integration pack, read documentation on said integration pack and in general was available for questions from the primary developer. The shadow did not program with the primary developer, as far as I observed. Company C used a few practices of agile methods such as stand up meetings, story cards and iteration planning.

6.1.2 Format of the Observations

The format of the observations remained approximately the same across the organizations but the schedule was modified slightly depending on the specific organization. Company B differed the most. During all observations I took handwritten notes.

Company A invited me to their organization a day early to meet the staff and familiarize myself with the organization. I observed three developers and spent three days with each developer. The first day with each developer was spent as a silent observer of how the developer worked in formal and informal meetings. The second day with each developer was spent asking questions periodically to understand more context of each situation I was observing. The third day was spent the same as the second day. I tried a think-aloud protocol [Ericsson93] but abandoned it once I quickly got the impression that I was "over staying my welcome" with the observed developer.

The format of Company B was slightly different. Because of the emphasis on group work and the open work environment, I spent all 9 days in the open work environment periodically switching among 6 developers and 1 customer representative. I made certain I spent at least one day and at most 2 days focused on one developer. I spent the first day at the organization as a silent observer and then varied the extent to which I asked questions, depending on the developer I was observing, their openness to conversation and the perceived severity of the task.

My experience at Company C was similar in format to Company A. I spent one day of introductions to the staff members and two days with each developer. I observed 4

developers. The first day with each developer was spent as a silent observer and the second day was spent asking questions periodically to understand the situation.

6.2 Method of Analysis

Analysis occurred via a three step process. The first step was the use of content analysis; the second step was building explanatory case studies; the third step was performing a cross-case analysis. The field notes written during the interviews were typed and analysis occurred on these field notes.

6.2.1 Content Analysis

Content analysis was used again to place paragraphs of the field notes into codes [Kripend80]. The approach to content analysis taken in this study was different than the approach taken in the first empirical study, in four ways. First, the dictionary used was pre-defined and was based on components of real options theory [Lasher06, Sullivan99]. Second, the codes were applied to a broader window than the window used in any of the coding in the first study. Third, significantly fewer codes were used (i.e. 5 instead of approximately 40). Last, the approach to validating the coding was different than performed in the first empirical study, both of which will be addressed in Chapter 11.

I coded sentences and paragraphs of the observation notes using the five parameters of real options analysis. The five parameters defined in real options analysis are primarily based on money or time. These values are numerical and are thus easier to evaluate and measure objectively. However, in order to apply these parameters to the observations of software design, I looked for *conceptual* examples of the five parameters of real options analysis.

The first parameter of real options analysis is the underlying value of the risky asset [Lasher06, Sullivan99) and I searched and coded the field notes for indicators of how designers valued the software design in which they made design decisions. This code was named **Value of Design**. The second parameter of real options analysis is the exercise price of the option [Sullivan99]. I searched the data for indicators of how costs were allocated to implementing a design change. This code was named **Cost of Change**. The third parameter

of real options analysis is the standard deviation of the underlying risky asset, including uncertainty or risk [Sullivan99]. I thus searched the data for what design uncertainties existed in the design change. This code was named **Design Uncertainty**. The fourth parameter of real options analysis is the time to expiration of the option [Sullivan99]. I searched the data for indicators of delaying a design change, and this code was called **Timeframe Established**. The last parameter of real options analysis is the risk free rate of interest over the life of the option. I searched the data for indicators of gains while waiting to make a decision and named this code **Gains During Timeframe**. Once again, I used content analysis because it offered a systematic and reliable approach to condensing text into codes that can be analyzed.

6.2.2 Explanatory Case Studies

After performing content analysis on the field notes I built a case study summary of the decision events observed at each organization. This second multi-case study consisted of 28 explanatory case studies across the three software organizations. The observations resulted in 12, 11, and 5 decision case studies from Companies A, B and C respectively. I examined each case study as an explanatory case study of the circumstances surrounding one design decision. I say the case studies were explanatory because I searched for relationships between the way that software designers made design decisions, and components of real options analysis [Yin02]. This was the same approach used in the first empirical study. Using knowledge of real options analysis I generated small ideas about each case, interpreting the case as it related to the attributes of real options analysis. I developed three interpretations of the case studies, addressing the Components of the Decision Making Process, the Goal of the Decision, and the Representation of Value in the decision.

The underlying concepts in these interpretations were based on concepts similar to the first study. First, objective and subjective views of design were examined, similar to addressing cues in the first study [Zannier07b]. One of the first parameters of ROA is the value of the underlying asset. I examined the data for any indicator of how software designers valued the design, whether it was in the way they spoke, (e.g. “this is good”), a dollar value placed on design (e.g. “this feature is worth *x* amount of dollars”), or how they associated software

design with some component of their work (e.g. a design task is a new feature a designer has to complete). Here, value did not need to be something quantifiable, or even a metric, it could be the way the design was represented, its role, for the designer, in the entire work environment. Second, how the problem was understood, via external factors such as marketing pressure or via a personal understanding of the problem were examined, similar to addressing problem structuring and external goals in the first study [Zannier07b]. Third, time pressure and delaying a decision were examined, similar to time pressure in the first study [Zannier07b].

Interpretation 1, Representation of Value: If designers value design in an objective fashion, then decision making was consistent with real options analysis in the value of an asset. If designers value design in a subjective fashion, design decision making was considered different than and inconsistent with real options analysis. Similarly, if the cost of change is defined in an objective fashion, the decision making was consistent with real options analysis in the cost of change. If the cost of change is defined in a subjective fashion, design decision making was considered different than and inconsistent with real options analysis.

Interpretation 2, Goal of the Decision: If decisions were made while considering external factors such as marketability of the product or "maximizing shareholder value" analogues, then the goal of the decision was aligned with real options analysis [Sullivan199]. However, if uncertainty, risk or volatility of the design to be changed lies primarily in "internal" factors such as a need to learn some aspect of the problem, then the primary goal is to reconcile this uncertainty through learning – a goal that sharply differs from real options analysis.

Interpretation 3, Components of Decision Making Process: If software designers were found to delay decision making and recognize the gains of delaying a decision then real options analysis was considered relevant to software design decision making. If there was limited evidence of delaying a decision or recognizing the gains of delaying a decision, and/or factors inhibited the delay and the recognition, then real options analysis was

considered less relevant to software design decision making and designers were seen not to be investing in flexibility, as per real options analysis [Erdogmus03].

6.2.3 Cross Case Analysis

The third step of analysis was to compare the results between cases. Using knowledge of design decision making from the first study [Zannier07] and knowledge of real options analysis, I generated explanations about each case study and compared them against other cases. I continuously refined these explanations by continuously incorporating more case studies, until the theory explained all the case studies. This theory became the description of design decision making via real options analysis.

6.3 Experiences with Qualitative Inquiry

There were aspects of this study that changed in light of my experiences with the first study. Before each study I knew the approach I would take to data collection (e.g. interviews, observations) and that I would code transcripts of interviews or field notes. However, the way the coding changed was not planned.

6.3.1 Changes from First Study

First, as mentioned in Section 6.2.1 the coding changed in four ways, in the second study. There were two reasons for these changes. The first reason was the significant time I invested in low-level coding (e.g. Small Window Coding) with little return on the investment. The SWC took up to 2 full days to complete, on a one-hour interview. However, the SWC removed much of the context of an interview and thus, from a qualitative, explanation building perspective, it was of less value than higher level coding (e.g. LWC, SRC, GRC). While the SWC facilitated validity checks, low-level coding was an extremely rigorous process for its role in conducting and publishing qualitative research.

The second reason for the change in coding was to address a recurring issue in feedback from reviews from publications. The interactive dictionary used in the first study was entirely descriptive but the results left reviewers asking the “so what?” question. After reading articles prescribing an approach to decision making [Sullivan99, Boehm00] I

thought a comparison between a prescription for decision making and a description of decision making would address the "so what?" question explicitly. Such a comparison would provide feedback about the suitability of the prescribed approach to decision making. Performing this comparison required a change in the codes used. Interestingly this facilitated one of the new aspects of this second study: the approach to validity.

6.3.2 Lessons Learned in the Second Study

I have identified three strengths and two weaknesses of this study. First, for the strengths, the observations were as "information rich" as they were "supposed" to be [Patton02]. Observing software developers working was extremely informative and allowed me to compare what had been said during the interviews with what was performed on a daily basis. Second, my experience with coding and explanation building matured through the second study. Third, I tried a different approach to coding that used external coders (after changing the codes that I was using as mentioned in Section 6.3.1). I reduced the number of codes used to reduce the cognitive load for the external coders who were not extremely familiar with the original set of codes (see Appendix B). This meant there was not as steep a learning curve for the external coders. The external coding was seemingly more acceptable to the larger software community. The weaker points of this study were simply unavoidable. First, during the observations I was faced with the challenge of a steep learning curve while I was observing: I had to understand numerous software problems within a short period of time and try to make the participants as comfortable as possibly during my observations. Second, regardless of my attempts, my presence had some impact on the participants. I address this in Chapter 11.

6.4 Chapter Summary

The second empirical study used observations at "real-world" software organizations to understand what aspects of real options theory occur in software design decision making. Software designers at three small software organizations were observed resulting in 28 design decision case studies. Analysis involved content analysis with a pre-defined dictionary, explanation building and cross case analysis. I now present the results of this study.

CHAPTER SEVEN: RESULTS OF AN EMPIRICAL STUDY of DESIGN DECISION MAKING via ON-LOOKER OBSERVATIONS:

Results from observations of software designers in small software organizations yielded explanations of the way software designers worked and themes related to real options analysis. These two components are evidence for the two main conclusions and set of eight smaller results presented throughout this thesis. The first main conclusion is that designers appropriate a design solution approximately as often as they strive for an optimal design solution, when making a design change. The second main conclusion is that the strength of a designer's mental model impacts the extent to which the designer considers alternatives. The set of smaller results from this study address agile environments in decision making, timeframes in design decisions and the applicability of real options analysis on software design decision making.

Throughout this chapter I use identifiers such as *CS2.5A_Q1* to tag an excerpt of a field note. The format of this identifier is as follows: The "CS" stands for "Case Study". The "A" (or "B" or "C") represents from which company the field note was generated. The 2.5 means the excerpt comes from the second empirical study and the fifth case study at the specified company. The "Q1" (or "Q2") represents the first excerpt (or second excerpt) taken from the case study. The excerpts are ordered for organizational purposes only.

In the field note excerpts I describe what a designer was doing (e.g. he looks at one section of code, he uses task manager). This type of field note came from the participant focusing on something specific in the source code by highlighting the code, by popping up a tool continuously (e.g. task manager), by talking out loud about what he was doing, and through my understanding of the situation when I asked questions like, "Can you tell me what you're doing now?"

7.1 Optimal Design Solutions?

The first conclusion that emerged from the data is that designers appropriate design solutions approximately as often as they strive for an optimal design solution, when making

a design change. I will show how this emerged through the codes **Value of Design** and **Cost of Change**, through three propositions built in analyzing the observations and through the interpretation from Section 6.2.2 entitled Representation of Value.

7.1.1 Value of Design

When coding using the pre-defined dictionary from the themes of real options analysis, I examined the field notes for the way in which software designers represented and discussed the design on which they were working. The code **Value of Design** was assigned to these sentences and paragraphs of the field notes. The value of the underlying asset (i.e. **Value of Design**) was represented primarily in two ways – either by a feature the design did not support, or by a bug it contained. A feature request was the motivator for a design change in 17 of the 28 case studies analyzed. That is, a design change occurred as a result of a new feature a designers had to complete, in 17 of the 28 case studies analyzed. Designers discussed the design in this way more than they discussed its structure, or following design heuristics, or the like. Excerpts of the field notes showing a feature request as the motivator for a design change are below.

I observed this developer when he was using the debugger to trace through source code. When he made a change to the code he talked about what he thought would happen given the change, then made the change, then ran the application to see if he was right. When he talked about what he thought would happen he was explaining the problem to me. It was not a formal verbal protocol.

CS2.3A_Q1: "[Designer has to] figure out format of data going between cashier's terminal and back to kitchen monitor because customer wants a feature. … He runs the system which has backend screen transmitting data to a front end screen and then the log file creates a record of everything that was sent over."

A senior developer assigned a development task to a developer.

CS2.5A_Q1: "[Customer] wants to have a coffee monitor, as well as Kitchen Video [feature]."

Below is a description of why the senior developer wanted this feature implemented.
CS2.5A_Q2: "Designer saw a feature for 10 seconds that a competitor has and also heard from one of the top dealers that customers had been asking for this feature in [their system]."

This developer also used the debugger and Edit-Find functionality to help him make a change to the code.
CS2.9A_Q1: "[Designer is] working on a timer for the coffee monitor, he has to have one timer for all 6 lists in the coffee monitor."

This developer said he had implemented a feature like this feature in the past.
CS2.10A_Q1: "[Designer] has to set the system so times stop when the order is done and has to remove items from the list."

A conversation between a senior developer and a developer ensued discussing this feature.
CS2.11A_Q1: "[Designer] has to do a replace function."

As part of a larger task this developer identified a feature the system required.
CS2.12A_Q1: "[Designer] has to make configuration form run once and only once."

This developer shown in the next two quotes had a functional user interface, but was unhappy with it. In watching him work he appeared very detail-oriented and truly concerned about the product on which he was working.
CS2.1B_Q1: "[Designer] wants to incorporate more workflow into the design."
CS2.3B_Q1: "[Designer] tries to create filtering on a form to not be redundant between a table and a wizard."

This developer did not understand the feature communicated to him and spent time discussing it and questioning it with members of the team, to achieve clarity.
CS2.5B_Q1: "[Designer] needs to provide more detail in user interface, a request that came from email from customer.

Two developers critiqued various ways to format a table, because they were unhappy with the existing format.
CS2.6B_Q1: "[Designers] have to determine how to list things in a table."

Two developers, pair programming, determined a solution to this problem quickly, based on past experience.
CS2.7B_Q1: "[Designer] has to determine how to order months using an ordinal."

Two developers and one customer representative discussed this feature briefly
CS2.8B_Q1: "[Designers] work from a notepad file that has a small list of requirements. They have to create a rule in the system, for a charge."

This was the most challenging feature the developers at Company B were working on while I conducted my observations. It was an on-going task.
CS2.9B_Q1: "[Designers have to] develop an algorithm to load beer onto a truck."

A large conversation about how to best accomplish this feature occurred between two developers and the customer representative.
CS2.11B_Q1: "[Designers have to] make adhoc rules be based on a built-in rule."

This developer had one solution to this feature but was not happy with it.
CS2.1C_Q1: "[Designer has to] figure out how to integrate with a system with no SDK."

This developer conducted two searches on the internet to help him in solving this feature request.
CS2.2C_Q1: "[Designer has to] make a message pop up to a user that an event occurred."

Feature requests came directly from the customer in 4 of the 17 cases where it motivated a design change. In 12 of the 17 cases, the feature request came from a developer as part of a

larger development task the developer was implementing. In 1 of the 17 cases a third party/partner company requested the feature.

A bug fix was the motivator for a design change in 10 of the 28 case studies analyzed. More than discussing design structure, following design heuristics or the like, the developers described the design in terms of its limitations – features that were not working correctly. Excerpts from the field notes, showing a bug fix as the motivator for a design change, are below.

On a Monday morning this developer's planned work changed when he received an email that a customer in another province found a bug when using the system. In the second of the following two quotes he described the situation flatly, as if the interruption was a normal occurrence.

CS2.1A_Q1: *"[Designer] said he is working on [another part of the system] and then periodically when a bug comes up he works on that bug. For example, this weekend [a customer] had a problem come up and he's trying to fix that."*

CS2.2A_Q1: "Email comes in, he reads it, says, 'So we had a problem out in British Columbia on the weekend and now I have to drop everything."

The senior developer focused on bug fixes when he had experience in the part of the system where the bug was, or when the bug affected many people, as shown in the next three excerpts.

CS2.4A_Q1: "There's a problem in the system so [the designer] is going through different versions of the code for a particular section of code comparing the different code."

CS2.7A_Q1: "Login and logout feature for employees has a bug and it's a bigger problem [than what he is currently working on]."

CS2.8A_Q1: "Before [the designer] left [the room] he said the decision to spend that much time on the problem has to do with the number of dealers who have complained. If it was just one store that had the problem there's no way he'd spend that much time on it. But lots of people were having the problem."

A group of six people, three developers, two customer representatives and a business owner who just listened, met quickly and informally when a customer representative was made aware of a problem in the system.
CS2.2B_Q1: "There are multiple bills per day for the same customer. [Two developers and one customer representative] are at office doors explaining the problem to [lead developer]."

A developer identified a bug in the way that time zones were handled. He identified the bug while starting to implement the following feature.
CS2.4B_Q1: "[Designer] scans code and recognizes that it handles only one order and they need it to handle more."

Initially this bug fix appeared simple, but one member of the pair programmers raised some concerns and recognized the fix was not a simple one.
CS2.10B_Q1: "[Two designers] are working together on a bunch of bug reports that they got over the weekend. Screen says, 'Bug 66 – Capitalization looks careless.'"

Two developers working independently found the solution to a bug by comparing each other's version of the source code.
CS2.3C_Q1: "[Designer1] comments that if the debugger is running the server will hang. [Designer1] directs [Designer2] to a properties dialog box and says to go to the "variance" variable. Then says, 'you don't have a variance variable.' [Designer1] directs [Designer2] to try something else and they recognize a problem in the system."

A feature request from the quality team led to this change.
CS2.5C_Q1: "[Designer has to] change a user interface to make inputs more constrained by putting a drop down box instead of a field."

In 3 of the 10 cases where bug fixes motivated design changes, other work was interrupted to fix the bug. In the remaining 7 cases, the bug was worked on as a normal flow of the

developer's day. It was selected from a bug report as a task for the day, or, in 3 of these 7 "normal flow" cases the developer spent the majority of his day handling incoming bugs.

The remaining case study (17/28 concerned feature requests, 10/28 concerned bug fixes) concerned selecting a technology to support web services and the value of the design was shown through a discussion of the advantages and disadvantages of web services. This discussion occurred between developers and management at a formal meeting. One of the 28 case studies indicated cost was also a way in which the design was represented (secondary to a feature request). An example excerpt from the field notes is below.

CS2.5A_Q3: "If [the software company] can make this part of the system then their customer pays [the company] $200.00 for software as opposed to paying [a partner company] approximately $1000.00 for hardware.

This excerpt is important because it is an outlier. It shows a quantitative, or objective representation of design. Arguably, this quantitative representation of design is just an example number, the system could have been valued at $250.00. However, I give the benefit of the doubt to a quantitative representation of design, for the sake of argument. For 27 out of 28 cases observed, design was discussed in terms of a new feature or a bug, which are both difficult to quantify. Requirements are often criticized for being abstract or vague, difficult to accurately specify. Bug reports can also be difficult to objectively evaluate and difficult to re-create. The subjective nature of design found in the observations conducted show a predominantly subjective Representation of the Value of software design. Thus, the first proposition of this study is as follows:

P1: Representation of the value of software design is predominantly framed in subjective terms.

From this proposition and similar to the first empirical study, I argue that at best, software designers make decisions that are boundedly optimal, because the bounds are subjectively defined.

7.1.2 Consequential Choice or Serial Evaluation?

The field notes that were summarized into 28 case studies were identified as decision events that used either consequential choice or serial evaluation. In 15 of the 28 case studies developers used consequential choice when resolving a design change. In the remaining 13 cases developers used serial evaluation. Table 7.1 shows the case study ID, a brief description of the motivator to the design change, and the approach to the decision. The case study ID is indicated with an A, B or C to identify the company, and a number of the order in which it is shown.

Company A used consequential choice in 4 of the 12 decision events observed (33.3%). Company B used consequential choice in 9 of the 11 decision events observed (81.8%). Company C used consequential choice in 2 of the 5 decision events observed (40%). The case studies show that in almost half the design decisions, alternatives were never considered. Serial evaluation was employed instead. An excerpt from the field notes showing the use of serial evaluation and satisficing (which is often found alongside serial evaluation [Klein98]) is below.

CS2.8A_Q2: From the help file [the designer] learned there are four different types of pings. … He tried the second type. When I asked why he picked that one he said, ‘’cause it was next on the list.’. … We went to [the testing] room and tested [his change] by logging onto Terminals 1, 2 & 4 with #3 unplugged, but [the test] didn’t pass. He reads a bit and says, ‘Unfortunately we’re doing everything the way the help file says to.’ He continues to read. Then says, ‘In an effort to save time, I’m going to do what I like to call a band-aid. … Even if it doesn’t totally fix it, it’s not perfect, it’s better than dying.’

This is a clear example of both serial evaluation and satisficing. The developer had a list of potential solutions and tried two of them, one at a time (serial evaluation). Next he implemented something that worked, even if it was not perfect (satisficing). There was no consequential choice. From this result, summarized in Table 7.1, I formulate the second proposition of this study.

Table 7.1: Quantitative Indicators of Consequential Choice in Agile Teams

Case ID	Motivator to Design Change	Approach to Decision
CS2.1A	Bug Fix	Serial Evaluation
CS2.2A	Bug Fix, if done, halts release.	Consequential Choice
CS2.3A	Feature Request	Serial Evaluation
CS2.4A	Bug Fix	Serial Evaluation
CS2.5A	Feature Request	Consequential Choice
CS2.6A	Feature Request	Serial Evaluation
CS2.7A	Bug Fix	Serial Evaluation
CS2.8A	Bug Fix	Serial Evaluation
CS2.9A	Feature Request	Serial Evaluation
CS2.10A	Feature Request	Consequential Choice
CS2.11A	Feature Request	Consequential Choice
CS2.12A	Feature Request	Serial Evaluation
CS2.1B	Feature Request	Consequential Choice
CS2.2B	Bug Fix, if done, halts release.	Consequential Choice
CS2.3B	Feature Request	Consequential Choice
CS2.4B	Bug Fix	Serial Evaluation
CS2.5B	Feature Request	Consequential Choice
CS2.6B	Feature Request	Consequential Choice
CS2.7B	Feature Request	Serial Evaluation
CS2.8B	Feature Request	Consequential Choice
CS2.9B	Feature Request	Consequential Choice
CS2.10B	Bug Fix	Consequential Choice
CS2.11B	Feature Request	Consequential Choice
CS2.1C	Feature Request	Consequential Choice
CS2.2C	Feature Request	Serial Evaluation
CS2.3C	Bug Fix	Serial Evaluation
CS2.4C	Chose Programming Language	Consequential Choice
CS2.5C	Bug Fix	Serial Evaluation

P2: developers frequently evaluate one alternative at a time instead of considering alternatives in their design decision making.

I argue that designers who do not compare alternatives do not strive for (even) a boundedly optimal design solution. The prevalence of serial evaluation in the observations, and the association between serial evaluation and satisficing are evidence for the conclusion that

software designers appropriate a solution to a design problem approximately as often as they strive for optimal or boundedly optimal design solutions.

7.1.3 Representations of Value

I used the coding and the interpretations presented in Section 6.2.2 to build explanations of design decision making as it pertains to real options analysis. For the code **Value of Design** I used the interpretation presented in Section 6.2.2 under the heading Representation of Value. The results of Section 7.1.1 and proposition 1 strongly suggest that design is valued in a subjective fashion. Real options theory recommends a quantitative value of the option under consideration [Lasher06]. Thus, design decision making is considered different than and inconsistent with real options analysis, from the interpretation entitled Representation of Value (Section 6.2.2).

The question remains, how does this pertain to the conclusion that software designers appropriate design solutions approximately as often as they strive for optimal or boundedly optimal design decisions? A goal of real options analysis is to strive to maximize shareholder wealth, or in other words, to select an alternative that optimizes (financial) value for those impacted by the decision. Given that real options analysis has been found to be inconsistent with design decision making, this pursuit of optimality found in real options analysis can be seen as inconsistent with software design decision making as described in the decision case studies presented. This is evidence for the conclusion that software designers appropriate design solutions approximately as often as they strive for an optimal or boundedly optimal design decision.

7.1.4 Summary of First Conclusion via Observations

The second empirical study used observations as evidence for to the argument that designers appropriate design solutions approximately as often as they strive for an optimal or boundedly optimal design solution. This section (Section 7.1) has contributed to this point by:

- Showing the code **Value of Design** as an indicator that software design is represented primarily in subjective terms, using excerpts from the field notes as data,
- Categorizing the case study summaries as employing consequential choice or serial evaluation, showing an almost equal occurrence of the two (15/28 case studies used consequential choice, 13/28 used serial evaluation),
- Using the interpretation found in Section 6.2.2 that shows a fundamental component of real options theory (specifying financial optimality) does not align with software design decision making.

7.2 Approach to Design Decision

The second conclusion that emerged from the data is that when making a design change, the approach a software designer uses is impacted by his/her understanding of the design problem. I used the strength of a mental model as a definition of understanding, which recognizes the cognitive aspect of design. To show that a designer's understanding of the design problem impacts the approach to decision making I will show data that was evidence that stronger mental models lead to serial evaluation [Klein98] and more open mental models lead to consequential choice [Luce58]. The evidence for this emerged in three ways: through the code **Cost of Change** and its resulting proposition, through the code **Design Uncertainty** and its resulting proposition, and through the interpretation from Section 6.2.2 entitled Goal of Decision.

7.2.1 Cost of Change

The pre-defined dictionary specified the exercise price as the second parameter required when conducting real options analysis, and it became relevant in examining software designers' understanding of design problems as provided by the case studies presented. I searched the field notes for discussions of design change and how that change was represented by designers. These discussions were found in two cognitive activities: mental modeling and mental simulation.

In numerous design situations designers stated what they thought would happen when they ran the system on which they were working. They compared this prediction with what did happen when they ran the system. Alternatively, designers ran the system and talked about what they thought was happening. This is seen as running a mental simulation and comparing it to the real execution of the system. In order to run a mental simulation one requires a mental model on which the simulation can run [Adelson85]. Twenty-two excerpts from the field notes are provided below, showing the use of mental modeling and mental simulation in bolded text, and examining developers working alone and in groups.

A bug was reported over the weekend and the developer tries to understand how it happened.

CS2.1A_Q2: [Designer] is talking to someone on the phone about the way he resolved the problem. 'I've started the terminal and restarted it and restarted it…' and he describes what happens. Listens to [person on the phone]. Says, 'you don't have to actually check if it does it automatically?' Listens more. Says this particular store has had a lot of problems with their database and power supply. … ***'Well I don't know if something happened like they ran an update or something happened that triggered their system.' … 'Why would it happen only on this one terminal, is what I can't figure out****. … So you can use mutex [to resolve the problem] and that's what I did.'*

In the following two excerpts, the developer tries to understand the flow of activity in a system, leading up to a bug.

CS2.2A_Q2: [Designer] is talking on the phone about a bug in the system. 'How did you get the end of day markers for Terminal 2? Oh, because it's automated.' He listens on phone. ***'It almost has to be an automate per terminal****. 10 messages in chronological order. … 'Web polling reported a corrupted database when it was trying to access a terminal.' … 'So you're getting the delete anti-marker. Isn't that an after the fact?* ***Delete anti would be an after the order. So you're getting delete anti markers? …. So its happening 2 hours after the fact.'… 'So are you talking about the date that's in the database record****?' Listens. 'Not the date time of when you received it. I wonder if the date time is wrong.'*

CS2.2A_Q3: [Designer] calls a third party company for a question about how their source code works. 'I have a question.' Asks [person on the phone] about getting an end shift marker and not having a start shift marker. ***'It just goes out into Neverland? You don't create the start shift marker****?' he asks somewhat skeptically.*

When trying to resolve the bug the developer thinks about scenarios he has to support in the source code, as shown in the following two quotes.

*CS2.2A_Q4: [Designer is] talking about code and how it gets like spaghetti code. 'I think [the problem] just got a bit more cloudy.****' He just thought of another scenario that could cause a problem****. He said he knows what his code does and was thinking of the progression of things.* ***'No, it's just a scenario I thought of,****' he said when I asked him if he saw something in the code that made him think of [the problem getting more cloudy]. …* ***'I'm thinking [the ID] could be overwritten.'***

CS2.2A_Q5: [Designer1] describes problem. [Designer2] makes a suggestion. [Designer1] says 'but what [third party company] is saying…' and describes the issue with that the third party company has. ***[Designer2] proposes a scenario****. [Designer1] says, "but this, but that". [Designer2] says, 'Oh, I see'. … [Designer1] asks about [part of the system].* ***They run some scenarios together. … They talk about parameters in the code****. [Designer1] says,* ***'we may need to test it, when you do a logout' He runs a scenario."***

Often the developers ran the actual system to see if they were right in their understanding of the system. This is shown in the next eight excerpts.

CS2.3A_Q2: [Designer] switches between code and log file, says, 'log file tells me there's a problem. He can tell system is not getting into an if statement cause log file doesn't have sentence that is inside if statement. ***Periodically runs system to test****…. Needs to see why [program] is not going into a nested if statement. 'So now that the log files aren't helping me I'm just going to run it using debug on the code'. … Uses a watch on a variable with the debugger and sees that the item he wanted to print was giving a subscript out of range error.* ***'That's why,' he says. Makes a change in code, re-runs but gets the same problem. Tries using index of 1 instead of 0 and re-runs [program]. 'Looks like it's a 1-based array. So it starts at 1 and not 0,' he says after it runs okay****.*

CS2.3A_Q3: Puts a message box in a function called UnLoad and got the messagebox popping up so said, ***'Something in my UnLoad is causing it?****' He went to UnLoad function and read it, said it was fine, then paused and said,* ***'oh oh oh oh oh. I think I know what it is.' Put a breakpoint at UnLoad function and re-ran system.*** *… He said he found [the problem] by pure coincidence.*

*CS2.7A_Q2: [Designer] spends time reading code, going to Microsoft Access database as needed, switching among functions, to remind himself of what the code says****. Read through gave him an idea, he tests the system with sample data.***

CS2.8A_Q3: [Designer] goes back to code and looks for places where sendOrder function gets called to see if anything triggers the problem. … Didn't find anything from looking at code so goes to [testing] room to simulate a scenario. … After a lot of trial and error [the problem] ended up being a 9 minute time delay. ***There was no real rhyme or reason to the way of testing things. He just went off logic developed from seeing what happened each time he tried something. He eliminated possibilities by running different scenarios*** *with control variables.*

CS2.9A_Q2: [Designer] … says he is writing out a possible solution on his paper. Goes to windows explorer and creates a backup [of the code]. Now he is going to change the code. Adds some code using lookahead function and 2 variables he created before. … Uses debugger to check the value of the CounterArray object. ***'Its always zero, that's why the time is not working,' he says. … Changes some code and runs system. Gets error when clicks Send button a 2nd time. I.e. timer is still not working.***

CS2.10A_Q2: Changes code and says ***he thinks it will work, so 'let's try.' It works. He says, 'because*** *I removed that Order, so now every Order I add is unique.*

CS2.11B_Q2: 'Essentially what we're going to do is create a rule dimension,' [Designer1] says. Then [Designer1] says, 'let's think of ***another way.****' He grabs a scrap paper and writes out [a tree hierarchy of objects]. They debate how many levels they have in a hierarchy and for adhoc rules you can have a lot of levels, for built-in rules you only have 3. They say, 'I don't know' or 'I'm not sure' when* ***talking about the problem and they rationalize it with each other. "Cause if you have this" type of conversations. … [Designer2] brings up a scenario****. [Designer1] says good point. [Customer Representative] says, 'I have a feeling we're going to have to change this. You have that*

same feeling,' he says noticing agreement in the designers. ***'It's a little hard to guess what they need when we don't have their data.' [Customer Representative] says. They talk about scenarios again.***

CS2.2C_Q2: [Designer working on a new feature has troubles with it]. 'And then something just hit me,' he says and explains that they have a different path format for window paths. He highlights the windows path … and says, ***'which brings something to mind.' He goes and tries it, it doesn't work. He shakes his fist jokingly and says, 'Damn you. So much for that idea.'***

When working in pairs or small groups developers proposed small scenarios of the way the system ran or the way a user would access the system. They communicated about the design problem using these scenarios. This is shown in the next two quotes.

*CS2.5A_Q4: [Designer1] provides possible solution and describes it. [Designer2****] asks a follow up question using a possible scenario. They work under this scenario****. [Designer1] picks up on it to ask a question.*

CS2.11A_Q2: [Two designers] talk about what the format of the message should be, they don't completely agree. [One designer] asks, ***'can we go through a real example?' [Other designer] opens whiteboard. They go through a scenario on the board.***

This developer explicitly referenced the mental simulations he ran.

CS2.1B_Q2: [Designer] had an idea for a table format, puts an example in Visio. Said he tossed around different ideas, thought about a tree structure, ***'but there are going to be a lot of orders.' … He says he's just going through ideas as to how to flag certain columns of the table. Just said he's thinking about putting a 1 in the run column, if it's a 0 don't run, if it's a 1, do something.***

Mental models were formed from artifacts of the customer domain, and this developer referenced these artifacts as needed, and discussed objects of the model represented in the artifacts.

CS2.3B_Q2: He goes and gets ***examples of sheets the customer works with currently but says for this particular problem he doesn't think he is going to show too much more*** *[on*

the user interface]. 'I'm just trying to think of what fields [the customer] needs to see.' … He says they need a notes field. … He makes a list saying customer currently prints by cases, kegs and floor and makes those items in his list. Then says the terminology is really weird and he describes why. ***Looks in notebook and goes back to screen and describes that the current design doesn't seem like the solution to him. For example floor is not the same as cases and kegs*** *[these are objects in the customer's domain that need to be represented in the software product under development]. He could design according to what they currently have which he knows will work but he thinks there is a better way that reflects a better understanding of the business.*

The next two excerpts show a certain degree of "rambling" when trying to understand a design problem. Ideas about the system were vocalized in snippets of speech that show some mental model existed.

CS2.4B_Q2: Goes through database and talks out loud to himself about time zones. ***'I didn't change the day. This might be a problem with our time zones. Why is it rewinding 4 hours back? This might be a bug with the time zone system'****. … He searches the code and says he didn't know they created an Allocation Record for a test but they do so he needs to verify that.* ***'If this passes it's purely by luck.'*** *Runs test and it passes, looks at code.*

*CS2.5B_Q2: [Designer] receives an email specifying a new feature request. Designer] looks at some XML and laughs, "custOrderNo, hostOrderNo, and orderNo****, I don't even know what these are. I'm going to change [the code] on the basis that I know what [the email] means. It must be order number. Ya, that makes more sense.'***

The next two quotes show, when developers worked together, not only did they discuss scenarios as presented previously, they discussed explicit components of the model.

CS2.6B_Q2: "[Designer1] tells me they're looking at code from another project with another customer to see how they did something more complex but same sort of problem.… We can just take this [code from previous project]. Just change the table.' [Designer2] says. They look through what they need and talk out loud about what they need. They read a large section of code and talk about changes. … [Designer1] says, 'How else are we

going to do it?'… They're looking at how to sort days and months. ... Before they've just left it but ***'it's a bit silly,' they say****.…[Designer3] comes over and says it's ok to put the number before it, like 01Jan. [Designer2]* ***laughs like he's not impressed****. … [Designer3]* ***agrees [to a suggestion*** *from [Designer1] but also says they've shown it to the customer in one way so if they can stay somewhat consistent with that it'd be good. … [Designer1] says* ***"Each row is going to be a customer, month, no, … each row will be customer by day****." He explains the "no" to [Designer2]. [Designer1]* ***raises a question****, if you ship an order over 2 days, it'll double count. [Designer2] asks "That happens?" [Designer1] says yes, [Designer1] says distinct count then. [Designer1] says need a month dimension. Create a fact table, it'll be a bit slower, [Designer1] says as an idea. …[Designer2] proposes a solution … and [Designer1] agrees."*

CS2.8B_Q2: [Designer1] starts explaining something to [Designer2] and ***[Designer2] fills in the rest like he knows what [Designer1] is talking about.*** *… They change a method that takes in a finite number of variables to take in a dynamic array. … They run the test and it fails. [Designer1] makes a code change and Alfred tells him to change another file.* ***'[The failing test is] probably because we had rows that can't be found,' [Designer1] says. [Designer2] confirms that he understands.***

Artefacts of the customer domain had to be verified with ideas that the customer had and what actually occurred, showing numerous models of the system.

CS2.9B_Q2: They had a few meetings with the [Customer] to sort through things. First the [Customer] said to order by one ID but it turned out not to be that simple. [Customer Representative] has an order form like what the [Customer] would have. It has Rack Number, EQ Number, Brand, Size, Quantity. The customer first said to go by Rack Number but then said that's also dependent upon the EQ Number ***and [the Designers] should just follow the order form that [Customer Representative] has. That ended up not being true either, which [Customer Representative] and [Designer] figured out from inconsistencies between values on the form representing weight and the customer telling them how they distributed weight****.*

These 22 excerpts show a comparison between the running of the actual system and a designer's thoughts about what might happen or has happened when the system runs. This occurred when designers were making changes to the design. That designers talked about scenarios by themselves and discussed scenarios with each other shows the execution of mental simulation. Explicit discussions of components of the customer's domain show mental modeling. I reach my third proposition of this study.

P3: Designers use mental models and mental simulations when making design decisions.

This is evidence for the second conclusion by showing the *use* of mental models and mental simulations. What remains to be shown is the strength of a mental model impacting the approach to design decision making.

7.2.2 Design Uncertainty

The predefined dictionary specified the standard deviation of the underlying risky asset as the third parameter required when conducting real options analysis. This concept was difficult to decipher in the field notes. I searched the field notes for any uncertainties discussed during the design change and while uncertainty was found, it was not identical to uncertainty as is defined in real options analysis.

I distinguish between two types of uncertainty found in the field notes; market uncertainty and structural uncertainty. I define uncertainty as a degree of knowledge that is deemed insufficient in some component of the represented concept, potentially introducing risk into the situation that uses that knowledge. I define structural knowledge as tacit or explicit knowledge where the quantity and representation are controlled in large part by the knower. An example is an author of a story shapes the contents of that story. I define market knowledge as tacit or explicit knowledge where the quantity and representation are controlled by someone other than the knower, and which impacts the knower's structural knowledge where the knowledge domains intersect. An example is demographic pressure that sales of book increase when happy endings are provided.

For software design I define structural uncertainty as the uncertainty in *internal* components of software design, such as the developers' knowledge of the system. I generated this term from the idea of problem structuring [Simon73, Guindon90a]. A software designer needs to understand (or structure) a design problem in order to solve it [Simon73, Guindon90a], relying on his/her intellect, skills, experience and the like. For software design I define market uncertainty as the uncertainty in *external* components of software design such as the customer requirements and changes to the requirements.

In the 28 decision case studies, 7 case studies contained structural uncertainty, 6 cases contained market uncertainty, 1 case showed uncertainty as a mix of the two types and for 14 cases it wasn't clear what uncertainty existed, if any at all. I will show indicators of structural and market uncertainty that lead to the fourth proposition of this study. The field note excerpts show signs of trying to understand the design problem, that is, structural uncertainty.

The first two excerpts show a developer trying to understand the flow of the system, when debugging.

CS2.2A_Q6: [The designer] tries to understand what is happening in the system when a bug is reported]. He is on the phone talking to someone about the problem and asks if that's the only range [the customer] is missing. 'Because that's the original order that caused the backlog in the first place'. … He writes down on a piece of paper what the [customer representative] says to him. 'Is [the system] still catching up now? Web polling reported a corrupted database when it was trying to access the terminal. … So you're getting the anti-delete marker. Isn't that after the fact? Delete anti would be after the fact."

CS2.3A_Q4: Looks in log file, reading and says, 'where does that come from?'. Looks through code, deletes a function, reruns, looks at task manager to see processes running. … Needs to see why [system] is not going into nested if statement.

The next four excerpts show developers admitting that their understanding of the system is not complete.

CS2.7A_Q3: [Designer] says he hasn't looked at the code in 2 years and never has enough comments so it could be tough to get into it.
CS2.8A_Q4: [Designer] is going to put in a check for a time stamp at the beginning [of code] and then a check to say if 5 seconds goes by (he picked 5 seconds randomly from the top of his head), he'll assume [the system] hasn't connected and he'll say it failed. 'That way I'll know. I think.' He says and pauses. 'Anyway, that's what I'm going to do.'
CS2.4B_Q3: Tries to print things out in an order. Says he doesn't know if its going to work 'cause it's a set, but he's going to try.
CS2.5B_Q3: He doesn't understand email [containing customer requirements], looks through code to understand but still doesn't totally understand. Has some reasonable ideas/understanding of what 2nd point in email means. … He now thinks from the 2nd email [from the customer representative that the customer representative] was talking about an order number which is different than the one he was thinking of. … He rolls back the changes because he thinks, 'we're having a bit of a communication issue.'

When working on a new feature this developer knew about the existing functions in the system that would be used.
CS2.9A_Q3: [Designer] says his biggest concern is that all lists have to use one function.

There is overlap in these excerpts with the excerpts for **Value of Change** because structural uncertainty is a part of shaping a mental model of a design problem (i.e. problem structuring and mental modeling are overlapping concepts [Guindon90a, Simon73]). The designers structured the design problem by eliminating the structural uncertainty they had about the problem, thereby shaping their mental model of the problem.

The market uncertainty shows uncertainty in design components that are beyond a designer's control. That is, rather than being an uncertainty in understanding, the uncertainty existed in a design component external to the designer's knowledge of the system. Such design components were user's reactions to the use of the system and customer requirements that were unclear.

In the excerpt below, a partner company places constraints on the development of Company A's system.

CS2.6B_Q3: A third party company wants part of [Company A's] software to include keystrokes so that a user can switch from [Company A's] software to [the third party company's] software. Right now [Company A's software] takes over the whole screen.

The next two excerpts show changing requirements after meeting with the customer.

CS2.1B_Q3: They originally thought they had to do orders for bars but through discussions with the customer they realized they needed [a feature] for other places. 'So the order component or the idea of an order became more prominent.'

CS2.2B_Q2: [Customer Representative] comes in and says from his meeting (conference call) [with the customer] they're going to have to make changes in the [customer's] project. … [Designer] asks if the customer is under the impression that they are getting a pre-built system or that it is customizable.

Uncertainty about what the customer wants is shown in the following three excerpts

CS2.3B_Q3: [Designer] said he is trying to figure out if the values are right so that the user can select them but he's worried that the values will be so frequently inaccurate that the customer won't use it at all.

CS2.9B_Q3: [Designer] talked to a customer on the phone who said basically you have to be [at the customer site] to see the design problem. … [Customer Representative] tells [Designer] to worry about bottles only right now, not cans. This is because the customer doesn't know how they want to organize the cans yet.

CS2.11B_Q3: [Customer Representative] says he's going to push for the customer to get their business down so [the designers] don't have to tweak [the system].

I do not argue that structural uncertainty is completely separate from market uncertainty and one case study exemplified this. The excerpt below shows knowledge about one instance of the system (i.e. structural uncertainty) can be inconsistent with knowledge about the customer's use of the system (i.e. market uncertainty).

CS2.1A_Q3: One of the terminals stopped working. [The Designer] describes the problem and says its code that is working in every other installation [of the system].

For the remaining 14 cases – half of the case studies - the uncertainty was not explicit enough in the field notes to assign a code and they are essentially outliers in the fourth proposition of this empirical study. These results show that uncertainty in software design exists in multiple forms, and this is the fourth proposition of this empirical study.

P4: Uncertainty in software design occurs in more forms than uncertainty in real options analysis.

Given this, I argue that examining a designer's understanding of a design problem (i.e. his/her mental model) merits examination in describing how decision making occurs. Thus, I searched the data for the strength of a mental model and the approach taken to make a decision. The case study summaries below show the strength of the mental model and the problem structures (defined or undefined) that the designer(s) had when making the decision. All approaches using serial evaluation are shown first.

In the first case study at Company A the developer had to fix a bug reported from a customer over the weekend.

CS2.1A_Q4: A bug fix comes in from over the weekend. Designer has to fix it so he does a series of looking on the internet, reading newsgroups, reading code and trying to contact someone. When he reaches a colleague he asks them about a past experience they had and what happened. When he works on code he copies and pastes code from the internet, changes it a bit and tries to run the system. He does this three times then calls a person at a third party company to let them know of the change he made. He says there was a similar situation before and they installed version 2.0 but that didn't work [this time]. 'So what I've done is I've modified web poling not to use that function. … you can use mutex and that's what I did.'

Problem structures in Strong Mental Model: Similar situation, bug in well structured system.

In the third case study at Company A, a feature request that the developer was working on led to debugging.

CS2.3_Q5: 'I think its because I was trying to do something I wasn't allowed to do. So I found a function on the internet.' [Designer] changes a line of code and reruns the system. He explains to me why he thinks it didn't work. "That didn't work either,' he says. Looks at a log file and says, 'Ya, that didn't work because its not supposed to be garbage like that. So that function is useless. … So I'm not going to use that anymore, I'm just going to write my own [function].'

Problem structures in Strong Mental Model: Rationalization of why system works and does not work, bug in well structured system.

A senior developer compares versions of code to fix a bug.

CS2.4A_Q2: [Designer] gets bug report and goes through 2 different versions of the same piece of code, comparing the code. He looks for the bug in the newer version of code that is not in the older version. He copies and pastes from the older version of code to the new one to resolve the problem.

Problem structures in Strong Mental Model: Existing well-structured system, bug in well structured system.

Pressure from an important partner company meant Company A would implement a feature.

CS2.6A_Q1: Even though they won't make any money off of this feature and it's a pain to implement they kind of have to do it, or feel like they should because [the partner company requesting the feature] is a big reason why Company A got into [an extremely profitable customer base].

Problem structures in Strong Mental Model: Pressure from third party company that provided them with business.

A senior developer at Company A fixes a bug without questioning a previous design decision.

CS2.7A_Q4: Read through of help file gave [Designer] an idea, he tests system with old sample data, but [then] … spends time finding useful data. … [Designer] found his solution, just to offer the user an option to proceed. He doesn't remember why he didn't allow the option in the first place.

Problem structures in Strong Mental Model: Help file, existing data, bug in well structured system.

This developer fixed a bug using satisficing and by trying solutions from a list of potential solutions.

CS2.8A_Q5: From the help file he learned that there were different types of ping 1)semtp 2)raw 3)udp 4)tcp. [Designer] decided to try the second option. When I asked why he picked raw to try he said, 'because it was next on the list. … He compiled code and put it on the main system. We went to the testing room and tested it by logging onto Terminals1, 2 & 4 with 3 unplugged but it didn't work. … [Designer] went back to help file. 'Unfortunately we're doing everything the way the help file says to.' … 'In an effort to save time, I'm going to do what I like to call a band-aid.'

Problem structures in Strong Mental Model: Save time, item next on the list, bug in well structured system.

This developer implemented a timer feature and rationalized his ideas and changes while he implemented his solution.

CS2.9A_Q4: Writes out a possible solution on paper and changes the code. Uses lookahead function and Edit-Find as he changes the code. Writes some code and runs system. It doesn't work so he uses the debugger, looks at help file and runs the system. Explains why he thinks its not working. Changes some code, copies and pastes code, tests it and it works. He explains to me why it worked.

Problem structures in Strong Mental Model: Rationalization of why system works and doesn't work, bug in existing well structured system.

In case study 2.12A the developer used previous code to save time when implementing a new feature.

CS2.12A_Q2: [Designer] says he's done this [design change] before and shows me the code he's copying and pasting . He says, 'its faster, why waste my time if I have already done it?

Problem structures in Strong Mental Model: Past experience.

This developer discovered a bug with how time zones were handled in the system. He worked through the code making changes as he had new ideas to fix the bug.

CS2.4B_Q4: [Designer] runs tests and it fails when he didn't expect it to, says timestamp is wrong and he says he doesn't really understand because he copied from another more granular test. [Designer continuously says what he thinks will happen, changes code and runs test to confirm.]

Problem structures in Strong Mental Model: Rationalization of why system works and doesn't work, bug in existing well structured system.

These pair programmers re-used a section of code they both remember suited their current feature request of ordering months.

CS2.7B_Q2: Two designers working together on a new feature and remember a way they had solved the problem a previous time. [Designer1] brings up the idea, what he remembers, [Designer2] says he remembers it too and they go to an API and search for the code. They read some text for a bit just skimming when [Designer2] starts saying he doesn't think the information they want is at that site. They look at the code, then go back to the site and [Designer1] says, 'here we go, that's what I wanted.' They copy code, paste it in their system, change it a bit to suit their needs, run it and it works.

Problem structures in Strong Mental Model: Previous experience, existing usable code.

This developer read some literature and came up with an idea for a new feature he was implementing.

CS2.2C_Q3: [Designer] has to implement a new feature to make a message pop up to say an event occurred. The designer reads up on instructions to understand how the system works and says he doesn't understand the error message in a log file. He says there are 2 sets of instructions, one doesn't apply and the other doesn't make sense. He goes to Google

to do a search, then an idea just hits him and he changes the code and runs it. It doesn't work. He talks about a part of the code, changes one piece of code and runs the system. He said he thinks it is working.

Problem structures in Strong Mental Model: Own idea, rationalization of the system.

Two developers came together to notice a difference in their versions of code that is the reason for a bug in one version.

CS2.3C_Q2: [Designer is reading code and is pretty frustrated because he doesn't know why a bug exists. He asks a colleague to run the system and he notices the colleagues' version of the code is different than his. He asks his colleague to run the system and realizes why the code isn't working, because of the difference between his code and his colleague's. He says he can fix it, changes the code and it works.

Problem structures in Strong Mental Model: Bug in well structured system.

A third party programming interface specifies the way in which a developer codes a new feature.

CS2.5C_Q2: [Designer] needs to change a field to a drop down box. He reads documentation on the API he is using, runs the system to see the existing user interface, says he needs to learn how forms work in a third party system, and says he knows how to make the change and is going to do so.

Problem structures in Strong Mental Model: Existing well structured working system, API documentation.

The following case study summaries show consequential choice. Notice the mental model is not as well defined as in the excerpts showing the use of serial evaluation.

Two developers debate ideas about how to fix a bug and how the fix will impact the next software release.

CS2.2A_Q7: A bug fix comes in and [Designer] looks at the code to understand the problem. He talks to people to figure it out as well but says he has to talk to [another designer], who knows the inner workings of the system, before he does anything. He waits

for the other designer and they go through some scenarios, going back and forth over the problem with each other, saying 'but this, but that' when challenging each other's ideas.
Few Problem structures in Open Mental Model: Talks to others to figure out problem, waits for person who knows most about system.

A senior developer gives a task to a junior developer and they discuss the ways in which the task could be implemented. The senior developer leads the conversation but the junior developer has some ideas as well.
CS2.5A_Q5: Two designers are working together on a problem, the first designer is giving the development problem to the second designer. The first designer gives the second designer constraints about the problem, like that it should be in .NET, there needs to be a numbering convention and it will work similar to a previous project they have done at the company. The second designer comes up with ideas about how to solve the design problem, and the first designer approves the ideas or turns them down. [Designer2] makes a suggestion but [Designer1] shoots it down and says, 'like a time function. … [Designer2] provides a possible scenario and describes it. [Designer1] asks a follow-up question using a possible scenario. They work under this scenario.
Few Problem structures in Open Mental Model: Second designer generates ideas that first designer approves or disapproves, run scenarios.

Again, these two developers (one senior, one junior) discuss how to implement a new feature.
CS2.11A_Q3: [Designer1] guides the conversation. There are two ways they have to decide between. Either recognize when you get a second object, or have another ID. [Designer1] says which option he would pick. [Designer2] questions [Designer1] on the option, then when his questions are resolved, agrees with [Designer1].
Few Problem structures in Open Mental Model: Designer 2 questions Designer 1 on the option.

A single developer with an attention to detail really thought about ideas for a user interface.

CS2.1B_Q4: [Designer] said he had different ideas, thought about a tree structure, 'but there are going to be a lot of orders.' Then he went on to explain that a table would be easier to sort. He opens a presentation file which they gave to [the customer] to show an understanding of the system. He had 2 alternatives for his design problem, both were technically feasible, but he points to one in the presentation file and says it was better for business reasons.

Few Problem structures in Open Mental Model: Had different ideas.

In this case study an informal conversation occurred when a large bug was identified and the developers were unsure how to proceed.

CS2.2B_Q3: A bug in the existing system starts a conversation among two customer representatives and three designers. They debate alternatives, and repeatedly challenge or turn down each others ideas and resolved that they need to talk to a partner company.

Few Problem structures in Open Mental Model: Need to learn more from partner company, turn down each other's ideas.

Again, the developer who paid attention to detail in case study 2.1B was concerned about the current way filtering was done on a user interface.

CS2.3B_Q4: He looks at two old ways they did filtering but thinks they could do it better. He describes his new way briefly.

Few Problem structures in Open Mental Model: Own idea.

This developer received a feature request that he did not understand and continuously challenged the idea with customer representatives.

CS2.5B_Q4: [Designer] gets an email from customer representative specifying a new feature request but he doesn't understand the wording of the email. He changes code on the assumption that he knows what the email means but then gets another email from the customer representative that confuses him more. He sends an email back to the customer representative to clarify. A second customer representative comes in and the designer and this customer representative discuss the problem. The designer doesn't understand why there are so many order number parameters and he discusses this with the second customer

representative. The second customer representative explains the problem as he understands it and the designer challenges him on some of the responses to the designer's questions. That conversation ends with the designer thanking the second customer representative. Later in the day though, when the designer goes back to the problem he still isn't clear about why they need so many order number parameters. The first customer representative arrives and explains the problem but the designer challenges him as well based on what he understands of the customer domain. The designer thinks the parameters are very redundant but ultimately does what the customer representative asks him to do.

Few Problem structures in Open Mental Model: Does not understand, continues to ask questions.

The next two case studies show two developers challenging each others ideas and recognizing that they do not know everything about the customer domain

CS2.6B_Q4: Two designers work on a feature request and try to figure out how to list things in a table. They look at code from another project to see how they did a similar but more complex problem. One designer suggests that they just do this new feature the same way, but the other designer doesn't like the idea very much. They say the old way, 'is a bit silly.' They debate new ways to do it an a customer representative joins their conversation. They go through different ideas, challenging each other or turning down each other's ideas. When they talk about the customer domain they don't have all the information. They continue to debate ideas and pick the alternative that is relatively simple, given what they understand of the problem at this time.

Few Problem structures in Open Mental Model: Turn down each other's ideas, do not have all the information.

CS2.8B_Q3: Two designers work on a new feature and start debating alternatives as soon as they being. They explain their ideas to each other, and sometimes they don't explain at all they just list an idea and when the other designer doesn't respond in a positive manner they don't continue explaining. They don't know everything about the customer domain and they ask their customer representative for clarification. He doesn't have all the answers either. The two designers pick an alternative based on what they understand about the

customer domain and talk briefly about a part of the system they could have impacted by making the design change they made.

Few Problem structures in Open Mental Model: Do not know about customer domain, customer representative does not have all the answers either.

This task was ongoing during my observations and involved numerous numbers of the development team conducting research on the task.

CS2.9B_Q4: For a new feature request that is a major part of the system, four members of the development team are involved. They are trying to develop an algorithm for storing beer on a truck that delivers to multiple locations. They have numerous meetings with the customer to see how a real truck is organized. They research different algorithms from what they learned in their undergraduate degrees and they compare the advantages and disadvantages of these algorithms. They even try going to a university professor for help. The problem is on-going and they discuss different alternatives with each other.

Few Problem structures in Open Mental Model: Research, help from a university professor, continued discussions with each other.

Two developers debated the significance of a change based on their respective understanding of the system.

CS2.10B_Q2: One designer tries to do a bug fix that he thinks is a simple task. He is in the middle of changing code when another designer comes to work with him. He explains what he is doing to the second designer and the second designer tells him to stop, that the bug is bigger than he initially thought and would require a change to the database. The first designer doesn't' agree at first and argues that if the tests pass he should be able to make the change. The second designer says he would feel more comfortable if they had end-to-end tests and further explains why he thinks the problem is bigger than the first designer originally thought. The first designer starts to understand and agrees. They make a note on the bug report, and don't change the system yet.

Few Problem structures in Open Mental Model: Designers do not agree.

Again, two developers debate design scenarios and recognize they do not understand the customer domain completely.

CS2.11B_Q4: Two designers are working on a new feature request. They immediately start considering alternatives to the problem and draw out a scenario on a scrap piece of paper. They say, 'I don't know' or 'I'm not sure" when talking about the problem and they rationalize it with each other. They bring up different scenarios saying 'cause if you have this.' They bring the customer representative into the conversation and ask him questions. He is only able to answer some questions, for many he just suggests to do what is most consistent with what they previously told the customer and whatever is easiest. He says he is going to push for the customer to get their business down. The designers resolve to do something that is technically feasible.

Few Problem structures in Open Mental Model: Say 'I don't know', rationalization, customer representative pushes customer to get business down.

This developer did some research into a problem and recommended the help of a consultant.

CS2.1C_Q2: Designer needs to interact with a system that has no SDK, which he has never done before. He emails someone to ask bout it. He knows there is a solution over SQL that he could use but that seems rudimentary to him. He waits to hear back about an SDK from someone, otherwise he is going to push for the company to get a consultant.

Few Problem structures in Open Mental Model: No experience, desire for a consultant.

The selection of a technology for web services was considered quite a bit by three developers.

CS2.4C_Q3: Three designers debate what environment for web services they should use: .Net or GSoap. The first designer asks what the limitations of each are and the other two developers explain the advantages and disadvantages of each technology. The first designer says they need management to approve the decision. … In a meeting with management the two designers explain why they think they should go with .Net. The first designer (of the three) says he agrees with the two designers. Management says he doesn't see it being a problem, to use .Net

Few Problem structures in an open Mental Model: Debate with each other, need management support.

Outliers. There was one outlier found in the 28 case studies. In case study CS2.10A, the observed developer developed a strong mental model by saying that he wanted to use an approach that used the least memory. He searched on the internet for the solution that used the least memory, comparing alternatives. This is listed as an outlier because it is the use of consequential choice with a strong mental model. The use of consequential choice was minimal, but it existed nonetheless.

CS2.10A_Q3: [Designer] looks up on the internet which data type uses how much memory. He says he wants to use the one that uses less memory.

These excerpts are evidence for the point that the approach to decision making is impacted by the designer's understanding of the system. That is, the stronger the mental model (i.e. the more s/he thought s/he understood the problem) the more serial evaluation was used. The more open the mental model (i.e. the less s/he thought s/he understood the problem) the more consequential choice was used.

Table 7.2 shows all cases, the problem structures they used in implementing a design change and their respective uses of serial evaluation or consequential choice. Outliers are in grey.

7.2.3 Real Options Analysis

I used the coding and interpretations of Section 6.2.2 to build explanations of design decision making as it pertains to real options analysis. For the code **Cost of Change** I used the interpretation in Section 6.2.2 called Representation of Value. Proposition 3 and the results of Section 7.2.1 show design change through mental models created by designers and the mental simulations that designers run on those mental models. These cognitive artifacts are extremely difficult to quantify because they are potentially impacted by experience, expertise and even personality. This makes design change difficult to

objectively define. Given this and the interpretation found in Section 6.2.2, I argue that design decision making is inconsistent with real options analysis, for the cases I observed.

Table 7.2 Summary of Decisions and Approach

Case ID	Problem structures in Decision	Approach to Decision
CS2.1A	Similar situation, bug in well structured system.	Serial Evaluation
CS2.2A	Talk to others.	Consequential Choice
CS2.3A	Rationalize why system does not work, but in well structured system.	Serial Evaluation
CS2.4A	Bug in well structured system.	Serial Evaluation
CS2.5A	Second designer generates ideas that first designer approves or disapproves.	Consequential Choice
CS2.6A	Pressure from 3rd party company that provided them with business.	Serial Evaluation
CS2.7A	Help file and existing data for bug in well structured system.	Serial Evaluation
CS2.8A	Save time, item next on the list.	Serial Evaluation
CS2.9A	Rationalize why system does not work, but in well structured system.	Serial Evaluation
CS2.10A	Search for best performance.	Consequential Choice
CS2.11A	Second designer questions first designer on options.	Consequential Choice
CS2.12A	Use past experience.	Serial Evaluation
CS2.1B	Designer had different ideas.	Consequential Choice
CS2.2B	Need to learn more from partner company.	Consequential Choice
CS2.3B	Own idea.	Consequential Choice
CS2.4B	Rationalize why system does not work, but in well structured system.	Serial Evaluation
CS2.5B	Designer does not understand, asks questions.	Consequential Choice
CS2.6B	Turn down each other's ideas, do not have all the information.	Consequential Choice
CS2.7B	Past experience, existing usable code.	Serial Evaluation
CS2.8B	Do not know customer domain.	Consequential Choice
CS2.9B	Research, help from a university professor, continued discussions.	Consequential Choice
CS2.10B	Designers do not agree.	Consequential Choice
CS2.11B	Say "I don't know", rationalize customer domain.	Consequential Choice
CS2.1C	No experience, desire for a consultant.	Consequential Choice
CS2.2C	Own idea, rationalization of the system.	Serial Evaluation
CS2.3C	Bug in well structured system.	Serial Evaluation
CS2.4C	Debate with each other, management support.	Consequential Choice
CS2.5C	Well structured system, documentation.	Serial Evaluation

The interpretation in Section 6.2.2 called Goal of the Decision says that real options analysis is aligned with software design decision making if design uncertainty occurs in external factors of the design problem. Proposition 4 formulated in these results states that uncertainty in software design occurs in more forms than uncertainty in real options analysis. The results of Section 7.2.2 also show that uncertainty in software design exists in structural and in market uncertainties through open mental models. Given this, the approach to managing uncertainty in software design decisions is inconsistent with the approach to managing uncertainty in real options analysis. That is, uncertainty in software design occurs in both external (market) and internal (structural) factors, but the uncertainty in real options analysis occurs in external (market) factors. Real options analysis does not provide for the uncertainty that occurs in software design decision making, specifically in internal (structural) factors. The uncertainty in structural factors is akin to a designer's mental model of the system and the data shows uncertainty in structural factors impacting the designer's approach to decision making. In this way the inconsistencies between real options analysis and design decision making are evidence for the point that the strength of a designer's mental model impacts the approach to decision making.

7.2.4 Summary of Second Conclusion via Observations

The second empirical study used observations as evidence for the point that the strength of a designer's mental model impacts the approach they take to design decision making. Section 7.2 has contributed to this point by:

- showing the code **Cost of Change** as an indicator that software design is subjective through mental modeling simulations, shown in excerpts of the field notes.
- showing that the strength of these mental models impacted the approach to decision making as shown in excerpts of the field notes. Strong mental models led to the use of serial evaluation and open mental models led to the use of consequential choice.
- Section 7.2.2 showing the emphasis of real options analysis on consequential choice in *market* uncertainty is inconsistent with design decision making.

7.3 Study-Specific Results

From the observations I formed impressions of the way software designers work and make decisions, which at the time could not be included in any general conclusion. I brought these impressions to the third study, looking for affirmation or contradiction of them. There were three results that emerged. The first was that agile environments led to the use of consequential choice more than non-agile environments. The second was that design decisions were not placed in timeframes or time boxes. The third was that real options analysis is an inaccurate description of software design decision making, for the decision cases that I observed. I will support each of these with results from the data.

7.3.1 Agile Environments and Consequential Choice

I compared consequential choice in agile environments versus non-agile environments to determine if consequential choice occurred more, less, or with equal frequency in small agile and non-agile software development organizations I observed. First I qualified the agility of each organization by determining the agile practices that each organization followed. This is found in Table 7.3. From this table it is clear that Company A did not follow any agile practices and Company B followed many agile practices.

Table 7.3: Agile Practice Present in Company

Agile Practice	Company A	Company B	Company C
Iteration Planning	No	Yes	Yes
Pair Programming	No	Yes	No
Versioning System	No	Yes	Unsure
Stand Up meeting	No	Yes	Yes
Unit Test	No	Yes	Yes
Unit Test First	No	Yes	No
Collective Code Ownership	No	Yes	No
Move People Around	No	Yes	No
Integrate Often	No	Yes	No
Refactoring	No	Yes	No

I was unclear about the agility of Company C. They claimed to be agile and used some of the agile practices as shown in Table 7.3 [Zannier07]. However, during the observations I did not observe traits I would expect of an agile organization. For example, I observed numerous developers who were *repeatedly* apologetic about interrupting work when asking

questions of their colleagues; I observed developers who were so focused on their *own* tasks they were unwilling to volunteer for new tasks or assist colleagues with new tasks, and I observed a separation of development tasks among team members to the extent that an integral member of the team calmly referred to their development team as a factory [Taylor07]. The general "feeling" of Company C was so significantly different than Company B (whose agility was extremely apparent) that I decided to report on the extent to which each company followed agile principles [Agile07]. Table 7.4 describes these observations, with supporting quotes for the areas where I saw disagreement between the claims of Company C and my observations.

The primary observation point was that when observing design decision making I observed more conversations in the small agile software team than in the two small non-agile software teams. This led to more use of consequential choice in the small agile software team than in the small non-agile software teams. Table 7.5 presents all the case studies showing the type of decision problem, the number of people involved in the decision and the approach to making a decision. Again the case study ID is indicated with an A, B or C to identify the company, and a number of the order in which it is shown.

First, Company A (non-agile) used consequential choice in 4 of the 12 decision events observed (33.3%). Company B (agile) used consequential choice in 9 of the 11 decision events observed (81.8%). Company C (self-claimed agile, observed non-agile) used consequential choice in 2 of the 5 decision events observed (40%).

I emphasize that consequential choice occurred when more than one designer contributed to the decision. In the 15 decision events where consequential choice was used, 14 of these involved more than 1 person contributing to the decision. In Company A more than 1 person was involved in decision making in 3 of the 12 decision events observed. In company B more than 1 person was involved in decision making in 9 of 11 decision events observed. In Company C more than 1 person was involved in decision making in 1 of the 5 decision events observed.

Table 7.4: Agile Principle Present in Company

Agile Principle	Present in Company A	Present in Company B	Present in Company C
Individuals & Interactions over Processes & Tools	Very Clear	Very Clear	**No** *O1: "[developer] says they've just started the shadowing idea, but …' it requires time and everyone is really busy. So it's the first thing to go.'"* *O2: "Assign big things to people and they go off to do individual planning. … An Iteration Planning Meeting ends. ... No one shared what tasks they had or that they needed help with anything. E.g. [developer] had her tasks on a piece of paper with yellow stick-its and they aren't on the [main] whiteboard. So those are tasks for her no one knows about."*
Working Software over Comprehensive Documentation	Very Clear	Clear	**No** *O3: Development team has a technical writer who writes all the documentation.* *O4:"During the meeting [developer] reminds people of the documentation process. [Manager] adds that they should use [tracking system] and should also communicate [with technical writer]."*
Customer Collaboration over Contract Negotiation	Unsure	Unsure	Unsure
Responding to Change over Following a Plan	Clear	Very Clear	**No** *O5: "[developer] … said, 'Everybody is expected to meet their schedule. There is no room to fall behind.'"* *O6: "'One more day [of research phase].'"*

Table 7.5: Quantitative Indicators of Consequential Choice in Agile Teams

Case ID	Motivator to Design Change	Number of developers	Approach to Decision
CS2.1A	Bug Fix	1	Serial Evaluation
CS2.2A	Bug Fix, if done, halts release.	2	Consequential Choice
CS2.3A	Feature Request	1	Serial Evaluation
CS2.4A	Bug Fix	1	Serial Evaluation
CS2.5A	Feature Request	2	Consequential Choice
CS2.6A	Feature Request	1	Serial Evaluation
CS2.7A	Bug Fix	1	Serial Evaluation
CS2.8A	Bug Fix	1	Serial Evaluation
CS2.9A	Feature Request	1	Serial Evaluation
CS2.10A	Feature Request	1	Consequential Choice
CS2.11A	Feature Request	2	Consequential Choice
CS2.12A	Feature Request	1	Serial Evaluation
CS2.1B	Feature Request	1	Consequential Choice
CS2.2B	Bug Fix, if done, halts release.	4	Consequential Choice
CS2.3B	Feature Request	1*	Consequential Choice
CS2.4B	Bug Fix	1**	Serial Evaluation
CS2.5B	Feature Request	1	Consequential Choice
CS2.6B	Feature Request	2	Consequential Choice
CS2.7B	Feature Request	2	Serial Evaluation
CS2.8B	Feature Request	2	Consequential Choice
CS2.9B	Feature Request	3	Consequential Choice
CS2.10B	Bug Fix	2	Consequential Choice
CS2.11B	Feature Request	2	Consequential Choice
CS2.1C	Feature Request	1	Consequential Choice
CS2.2C	Feature Request	1	Serial Evaluation
CS2.3C	Bug Fix	1	Serial Evaluation
CS2.4C	Chose Programming Language	4	Consequential Choice
CS2.5C	Bug Fix	1	Serial Evaluation

*developer asked entire room open question. **a 2nd developer helped part-way through task.

A contributor to this result comes from the first empirical study. Interview participant 22 was an outlier in the study, considering alternatives when he also had a strong understanding of the problem. However, the decision required agreement from more than one person, and the group spent a week considering alternatives. This provides an

explanation for interview participant 22 being an outlier in the first empirical study and is evidence for the observation in the second empirical study. As added support, interview participant 20 also considered an alternative to his design problem in light of commentary from another developer on the topic.

It is not my intent to say that consequential choice and multiple decision makers are related irrespective of the strength of the mental model. Mental models can be shared among team members [Walz93]. My observations at Company B showed members of a development team challenging the mental models of their teammates with questions and with their own mental models. This potentially opened an individual mental model, and provided opportunity to consider alternate mental models, until a shared mental model emerged from the team's discussion. In this way the idea of an open mental model is not a negative concept. Rather, it provides opportunity. The point is that more people involved in a design decision potentially leads to a temporarily open mental model and the use of consequential choice in order to strengthen a mental model shared among those involved in the decision making. The result is that, of the observed organizations, the organization that was most agile fostered an environment in which multiple developers naturally became involved in design decisions. Consequential choice to strengthen a shared mental model, followed.

7.3.2 Time-boxing Decisions

The fourth and fifth parameters of real options analysis are the time to expiration of an option and the risk free rate of interest over the life of an option, respectively. The codes to represent these concepts were **Timeframe Established** and **Gains During Timeframe**, respectively. I searched the data for indicators that designers delayed a decision (**Timeframe Established**) and discussed the gains or losses of delaying that decision (**Gains during Timeframe**).

Only 5 of the 28 case studies discussed delaying a decision. The delay was either to acquire more information from someone or because the decision wasn't considered important (e.g. grammar in the user interface). None of the case studies showed developers explicitly discussing gains while waiting to make a decision. When designers indirectly referred to

gains, the gains were not quantified. An example of delaying a decision is below. Three designers and two customer representatives discussed a request for a change to a requirement that came up late in the project's schedule. The group debated how to resolve the changing customer requirement and the resolution was to wait to discuss the changing requirement with a partner company. An excerpt of the field notes is below.

CS2.2B_Q4: [Designer1] doesn't think [a proposed idea] will resolve how much the customer is upset. [Designer2] summarizes the situation. [Designer1] makes a suggestion as a possible resolution but [Designer2] doesn't agree. They talk about another idea. [Designer1] asks, 'Do we want to do this or not, within this timeline? Because if so then they will have to push back the schedule.' [Customer Representative] says just wait until they find out what [a partner company] says.

Very few signs of delaying a decision and very few indicators of the gains made while delaying the decision existed, and thus, the fifth and sixth propositions of this study are as follows:

P5: Delaying a decision occurs infrequently in software design decision making.

P6: Clearly specifying gains while delaying a decision occur infrequently in software design decision making.

The infrequency of delayed decision making (proposition 5) makes it extremely difficult to map design decision making to real options analysis. A fundamental component of real options analysis is to wait to make a decision. The results show that designers rarely wait to make a decision while engaged in software development. The designers observed developed literally countless mental models and mental simulations of the systems they were developing but the results strongly suggest these mental models and mental simulations are not placed on some project timeline. That is, developers did not develop one mental model for the scenario where they made the decision immediately and another mental model where they recognize that a decision did not need to be made until a later project date, when they had collected more information.

The results on gains (or losses) while waiting to make a decision were similar. The few times that developers did wait to make a decision was either to glean information from someone or when a design aspect was unimportant. In these few cases, these gains were only briefly mentioned. They were not quantified, nor were they given much attention.

Given these results, the observation point is that software designers do not frequently use time-boxes when making design decisions.

7.3.3 Real Options Analysis

The third observation point was that real options analysis is an inaccurate description of software design decision making. The first proposition, that software design is represented primarily in subjective terms, shows that design decision making is inconsistent with real options analysis, given the interpretation in Section 6.2.2 entitled Representation of Value. The third proposition, that designers form mental models and mental simulations when making design changes also shows that design decision making is inconsistent with real options analysis, given the interpretation in Section 6.2.2 entitled Representation of Value. The second proposition, that designers frequently do not consider alternatives in software design decision making is a mismatch with the fundamental purpose of real options analysis: to provide an approach to considering alternatives. The fourth proposition, that uncertainty in software design occurs in more forms than uncertainty in real options analysis, shows the goal of decision making in real options analysis is inconsistent with the goal of software design decision making. Lastly, from propositions 5 and 6, there was limited evidence of delaying a decision or recognizing the gains of delaying a decision, real options analysis is considered less relevant to software design decision making and designers are not seen to be investing in flexibility, as per real options analysis [Lasher06]. Given the numerous inconsistencies between software design decision making and real options analysis, the observation point is that real options analysis is an inaccurate representation of software design decision making where design decision making is as described in the 28 case studies provided in this study.

7.4 Commentary on Real Options Analysis

Aligning Real Options Analysis with the design decisions that I observed was extremely challenging and motivates questions as to why I pursued this approach to evaluating design decision making. There are, however some advantages to this comparison, that suggest real options analysis is relevant to software design. First, I think these results emphasize a need mentioned in [Boehm00]: the software community needs a mapping between technical components such as a design heuristic or a design feature, and the business value that each component provides. I think the mismatch between real options analysis and the design decision cases that I observed would be facilitated by such a mapping. Second, real options analysis emphasizes the idea of flexibility, an important concept in software design. Whether it is flexibility provided by timeframes on a decision or flexibility provided by the way a design feature is implemented, I think the concept is important and real options analysis brings this to the forefront. My data strongly suggests that flexibility provided by timeframes on a decision needs work (i.e. it is not clear that this is a common practice), but the idea of last responsible moment [Poppend03] does not suggest that delaying a decision for flexibility is not applicable to software design.

If the software engineering community is going to move forward with using real options analysis, I think the business side of software development needs to become more involved, specifically in the area of developing mappings between technical components of design and the business value of software [Boehm00].

7.5 Chapter Summary

The results of the second empirical study are evidence to the two main conclusions and set of eight smaller results. The first main conclusion is that software designers appropriate a design solution approximately as frequently as they strive for an optimal or boundedly optimal design solution. This was shown through the code **Value of Design**, through general observations of the use of consequential choice and serial evaluation, and through the interpretation entitled Representation of Value. The second main conclusion is that the approach to a design decision is impacted by the strength of the designer's mental model. The stronger the mental model, the more serial evaluation is used. The more open the

mental model, the more consequential choice is used. This was shown through the codes **Cost of Change** and **Design Uncertainty**, and through the interpretations entitled Representation of Value and Goal of Decision. The first smaller result was that Agile environments led to an increased use of consequential choice by encouraging communication about the strength of individual mental models. The second smaller result was that software designers do not place design decisions into time-boxes. The third result was that real options analysis is an inaccurate description of software design decision making. Given these results from the second empirical study I present the third study.

CHAPTER EIGHT: PREPARATION FOR AN EMPIRICAL STUDY of DESIGN DECISION MAKING via PARTICIPATORY OBSERVATIONS :

The third empirical study consisted of participating in design decisions and analyzing a thick description of the environment in which the participation occurred. This case study consisted of participatory observations in a software development team of approximately 10 people. The participatory observations lasted 6 weeks. Analysis occurred via explanation building with a lens on explanation-based decision making. This study differs from the previous two studies in two ways. First, it does not use content analysis. Rather, it uses a "thick" case study description to provide as much context as possible. Second, the results of this empirical study produced two components that merit discussion. The first is the decisions that were made during the 6 weeks of participatory observations, as was examined in the previous two studies. The second is the use of action research [Schwandt01], to effect the way in which decisions were made in the software team. The results of the first component were compiled to build an explanation about design decision making, as it relates to explanation-based decision making [Penning93]. This chapter describes data collection via descriptions of participatory observations and my role in the software team. This chapter then describes data analysis performed. Last, this chapter describes my experiences with qualitative inquiry after this third empirical study. In this study a "case" or "unit of analysis" is a 6-week experience in a small software team discussing and executing design change.

8.1 Method of Data Collection

The participatory observations were conducted from May 4th to June 15th, 2006. The software team was originally contacted to participate in this research as part of empirical study #2. However, the software team was also looking to hire someone to track numerous decisions they had to make on an upcoming project. Thus, the idea to conduct participatory observations emerged as truly opportunistic sampling [Patton02, p240]. The company was originally contacted through a professional contact already working in the software team. Throughout this discussion I refer to the company at which I conducted participatory observations as E-Solutions Inc.. In addition, much of the discussions will use a first person

point of view in describing the work environment, because of my involvement with the software team. I provide a summary of the basic statistics of the software team at E-Solutions Inc. in Table 8.1.

8.1.1 Description of the Organization and Software Team

E-Solutions Inc. is a large power generation organization operating internationally. The software team within E-Solutions Inc. consisted of 18 people in total. Of these 18 people, 14 people were in some form of a development role. This role varied for each person. Some people were employees of E-Solutions Inc., performing software development within the company's existing software architecture. Some people were independent contractors hired for their knowledge of and access to software architectures that could extend the company's existing software architecture. Last, some were in technical administrative roles, (contractors or employees) servicing the company's underlying infrastructure. The specific titles of each of these people varied, but for the purposes of this study these people will all be referred to as designers, for three reasons. The first reason is for the purposes of anonymity. The second reason is to remain consistent with the first and second study. The third reason is that the project described consisted of numerous dependencies between each of the designers, regardless of specific role or title.

The remaining 4 people in the development team of 18 consisted of four different roles. One person was the manager of the project that will be described, and oversaw all meetings and decisions made on this project. One person was the project tracker and recorded information on project tasks for each designer. One person was in a senior development and senior technical administrator role, which meant his role consisted of a little bit of everything, including some project management. For the purposes of this discussion this role will be called senior designer. Finally, my role was to assist the senior designer in facilitating and tracking the decisions that would be made for the project that will be described.

Given that E-Solutions Inc. was a large organization not specifically invested in software development, they did not use agile methods. The software team observed in E-Solutions

Inc. had members who had experience with agile methods and a manager who knew a bit about agile methods, but agile methods were not used in the team. Development tasks were broken down and assigned to individuals. However, the environment of the software team was an open one. I was encouraged by the manager to ask questions of anyone in the team whenever I needed it. This occurred when I first started and needed to learn the existing architecture, and it occurred later in my time with the team, when some of my tasks changed to that of a developer. This communication existed among all the team members and I found everyone willing to be interrupted and help each other with their technical questions. In short, while development tasks were allocated to individuals and agile methods were not used, there was a "team feel" in the group.

Table 8.1: An Overview of Observed Software Companies

Parameter	**Company 1**
Breakdown of People Observed	1 Manager 1 Senior Designer 1 Decision Facilitator 10 Designers
Number of People Observed	13
Total Number of Designers in Team	14
Days Spent with Each Participant	0
Time Spent at Organization	6 weeks at ~85% (i.e. almost full time)
Role of Agile Methods in Organization	No Role
Division of Labour	Individual

8.1.2 Format of the Observations

The format of the observations was significantly different than those conducted in the second empirical study because I was working in the software team. I took handwritten notes to facilitate my understanding of technical concepts and to track my thoughts during my time with the team.

I was brought into the team to facilitate decision making on an upcoming project. The project was the launch of a new portal for E-Solutions Inc.. In order to make the decisions concerning the new portal launch, team meetings were held three times a week to discuss relevant issues. Four designers on the software team did not regularly attend the meetings because of other priorities, thus the number of designers I worked with during the meetings was 10 (n.b. not including the project manager, project tracker and senior designer). The

meetings were set for one hour. They initially occurred on Tuesday and Thursday of each week, then were moved (after the first week) to Monday, Wednesday and Friday mornings. The topics of the meetings were different aspects of the new portal launch. During the meetings and during the participatory observations I did not focus on any one designer to observe. Instead, I ran the meetings during the week, tracked the decisions that needed to be made regarding the new portal launch and participated in the decision making topics myself.

8.2 Methods of Analysis

Analysis occurred in three ways that differed from the first and second empirical studies. The first approach was to provide a thick description of the events that occurred at E-Solutions Inc. that acts as evidence for or against the two main conclusions presented in the first two empirical studies. The second approach was to compare the events that occurred at E-Solutions Inc. with an understanding of explanation-based decision making [Penning93]. The third approach was to compile a list of observations from the action research I conducted at E-Solutions Inc..

8.2.1 Thick Description

The first approach to analysis is to describe the events that occurred at E-Solutions Inc. to provide insight into the context in which decisions were made. “[T]o thickly describe social action is actually to begin to interpret it by recording the circumstances, meanings, intentions, strategies, motivations and so on that characterize a particular episode” [Schwandt01, p255]. The thick description does not present results with intermittent quotes or field notes excerpts as was done in the first empirically study (Chapters 4 & 5), and the second empirical study (Chapters 6 & 7). Rather, the thick description of the events that occurred at E-Solutions Inc. is a significant portion of the analysis and the results of this third empirical study (Chapters 8 & 9). To produce the thick description I used a summary of notes taken while conducting the participatory observations and my own narrative [Schwandt01, p168] of my time at E-Solutions Inc.. From this description the entire 6 weeks comprise a detailed case study of design decision making. General observations of the pursuit of optimality or bounded optimality, and general observations of the use of

consequential choice or serial evaluation in light of the presented design problems are taken from this case study as evidence for the two main conclusions of this research. In addition, much of the description of my time at E-Solutions Inc. provides a foundation for the smaller results of this experiment.

8.2.2 Explanation-Based Decision Making

The use of explanation-based decision making as a lens on the final study of this research was strongly motivated by the first two studies. After the first experiment the point that designers do not consistently strive for an optimal or boundedly optimal design was relatively salient. After the second experiment the issue of *how* software designers did pursue optimality or bounded optimality was intriguing. The informal conversations I observed in experiment 2 exposed the use of consequential choice and the potential to achieve an optimal or boundedly optimal design solution. Thus, the extent to which members of a design team conversed and the nature of that conversation interested me as I approached experiment 3. From my review of decision making approaches (found in Chapter 2) I believed explanation-based decision making to be the closest "fit" to the conversations I had observed at the three companies in experiment 2. I had this decision making approach in mind as I worked in the software team at E-Solutions Inc..

8.2.3 Affecting Decision Making

Analysis of decision making occurred during my time at E-Solutions Inc., because of my role as decision facilitator. Much of my notes from the participatory observations concerned attempts to positively affect design decision making, and my role in the team. I present excerpts of these notes verbatim and include narratives around these excerpts to provide a context for the lessons learned from this study. This is a component of the thick description.

8.3 Experiences with Qualitative Inquiry

There were three strengths and two weaknesses of this empirical study. The first strength of this study was that I gained personal experience with design decision making, which made the complexities of performing design tangible. The risk of theorizing about design

decisions with little practicality was significantly reduced. The second strength was the detail I could pursue on the topic, with individuals with whom I was working. One conversation in particular followed from my attempt to implement change, and shaped my understanding of issues that had an indirect relationship to design decision making. These were issues that ultimately impacted my understanding of explanation-based decision making in software design environments. If I had not been immersed in the work environment I am certain this conversation would not have occurred. The third strength of this study was the ability to validate my ideas with people outside a research environment. The results of this study were summarized and presented to some members of the development team at E-Solutions Inc., to determine their validity. Again, if I had not been immersed in the work environment and established relationships with these people, this validity would have been much more challenging to execute.

A first weakness of this study was the difficulty of learning pertinent information quickly in a complex work environment. Design meetings began immediately upon my arrival at E-Solutions Inc. and there was much to learn. This was of course, a necessary evil and an interesting challenge. The second weakness was my lack of experience in implementing change in an organizational team. The difficulty in doing this was impressive and sparked my interest in organizational change. Specific challenges will be presented through the thick description and results of impacting design decision making.

8.4 Chapter Summary

The third empirical study used participation in a “real-world” software development team to further understand how design decision making occurs, to help improve the way it occurs and to determine how explanation-based decision making aligns with an understanding of design decision making. I spent six weeks with a software team participating in design decisions for a new portal launch, resulting in a thick description of the decisions made. I now present the results of this study.

CHAPTER NINE: RESULTS OF AN EMPIRICAL STUDY of DESIGN DECISION MAKING via PARTICIPATORY OBSERVATIONS :

Results from participating in design decision making yielded the final contributions on the conclusions and smaller results of this research, as well as insight into impacting software design decision making. I contribute to the point that software designers appropriate design solutions approximately as often as they strive for an optimal or boundedly optimal decision and to the point that the strength of a designer's mental model impacts his/her approach to decision making. This study differs from the previous two studies in its third contribution, which is a recommendation for effecting change in software design decision making. Finally, I provide a set of results which addresses the action research conducted during this study and the strengths and weaknesses of the action research. These points are accomplished in 2 ways. The first is through the use of a thick description, as is provided in Section 9.1. The second is through a lens on explanation based decision making. This is the third of three studies that comprise this research and is the last basis for the cross case analysis provided in Chapter 10. In this study a "case" or "unit of analysis" is one 6-week experience at E-Solutions Inc.

9.1 A Case Study of Software Design Decision making at E-Solutions Inc.

Pat[1] met me in the lobby of E-Solutions Inc on my first day. It was Thursday, May 4, 2006. Through a set of locked doors, up the elevator, through another set of locked doors and around two rows of brightly lit cubicles I saw a familiar face to whom I owed my opportunity at E-Solutions Inc. Seema sat at her desk – a PC on her left and a laptop on her right. Pat greeted her loudly and she smiled in response. He asked her if she "had a minute" and she said of course. The three of us stepped into a large conference room, literally 4 feet from Seema's desk. Inside, Pat did most of the talking.

There is an existing portal right now for the company that is more of a website than a portal. Everyone can see everything. We want to separate it into 4 locations: Canada, United States, England, and South America. The 4 locations are where E-Solutions Inc. has

[1] Names of participants have been changed.

main offices, and each location is to have their own portal, sometime in the future. People across the company are to post to the portal, and we want to establish workflow so that they can post to the portal consistently and easily. We also want a consistent style and we want the people using the portal to be able to control the look and feel of their portal.

If I had any questions Seema would answer them, unless I had any immediately. I said I probably needed to understand a bit more and then would have dozens. Pat agreed, then mentioned a team status meeting at 9am, where I could meet everyone and get started. Later in the day I would meet specifically with Brian, the senior designer, and Pat again. I would be working with Brian on the portal project.

As Pat left the conference room Seema and I chatted. She said she was glad I was at E-Solutions Inc. and she immediately addressed the portal jam sessions, meetings the group had been having about the portal project. The structure of each meeting was, well, unstructured. The outcome of each meeting was vague. Last, Seema articulated another concern about the portal jam sessions.

*"And Carmen, I want to know **why** I am sitting here."* Seema pointed to an imaginary seat at an imaginary portal jam session, on the table in front of her.

The 9 am team status meeting was about to begin and we took the stairs to a conference room on the third floor. I met the majority of the developers involved in the portal jam session meetings: Brian, Greg, Mike, Ethan, Joe, Jeffery, Clayton, Ben, Bernard. Pat chaired the meeting.

Everyone should gather what they like and don't like about search, where are you going to search from, what do you expect to happen when you search, who is going to work on it? How are things with WebSeal Jeffery? A user going to the portal has to go through WebSeal, WebSeal can cache personal credentials. What about web services? XSLT converts Domino DXL. We can now surface or reference any domino document without having to jump to a new window.

Jeffery responded to Pat's questions, Joe responded to Pat's questions, Ben responded to Pat's questions and Greg asked some questions. Brian added responses to almost every question asked. The meeting ended, seemingly suddenly. I went back upstairs where I plugged in the laptop I brought with me. Like seven of the developers (over half the team), I was a contractor, not an employee of E-Solutions Inc. I opened a small binder I had been given, the title of which read, "Portal 2006". Inside, it read,

This year the portal will be re-branded and a new navigational structure will be created. As part of this effort there must be concentrated change management to help existing portal customers migrate to the new portal structure. This migration is an excellent opportunity to highlight the benefits and applied usage of the portal. … A critical success factor for the program is the adoption and utilization of the process and tools offered up via the portal…

I continued to read as Seema and Clayton discussed issues with a task they were working on together. In the background Pat, Brian, Ethan and Mike laughed at something Greg said. Later that afternoon I met with Pat and Brian about my role.

It is up to me and Brian to collaborate to facilitate the portal jam sessions. The team has been building up a lot of stock in terms of content – code, etcetera – that was never moved to production. These items could be used to help determine what goes into the new portal. The developers should come to the meetings prepared with what/why/why not, a one-pager of issues they have related to their tasks. Or get the developers to write a topic on the board and generate discussions around that, with action items from each meeting. Everyone should be involved in the discussions. They already had a few portal jam sessions and assigned tasks to people. We have till the end of May to make the decisions for the new portal.

Brian and I nodded and listened, and asked questions periodically, while Pat spoke. Pat then left and Brian and I chatted briefly about the next day. Tomorrow would be my first portal jam session.

My first portal jam session began with Brian saying to all of us,

"This is going to be a working session. To set up the ground rules of play." Brian has created a virtual room in an online tool called QuickPlace. The room has folders for development categories. Brian asked if, from Pat's discussion of things the developers were working on, did we need more categories? The categories are: Availability, Content Management, Hardware, Look & Feel, Network, Personalization, Security. A pause in the conversation. Brian says we need to create a form to capture the decisions we make. We need to document our design decisions so if we get challenged we can answer why we did things the way we did. Ethan says he's written comments in the code. Brian rhetorically asks if that would be enough for a high level discussion. A pause in the conversation. Brian says we need to decide what the form is going to look like, and we'll create the form in QuickPlace.

I was confused. I thought the meetings were supposed to provide an opportunity to make decisions, not make decisions about making decisions. If there was work to do to support the decision making process, wasn't that what Brian and I were supposed to do together? That is, I thought the meetings should occur as they naturally would. I suggested that maybe we could run through an example of some of the conversations they had been having about the portal so that I could see how things operated. Greg provided one example and I had some follow up questions, and a small discussion ensued. Shortly into the discussion Brian guided the conversation back to the components of the decision form and the discussion ended. Still baffled and noticing the meeting time was coming to a close, I volunteered to create the decision form. I said I would circulate among the team members today to discuss what they liked and disliked about the portal jam sessions, and create a decision form by Wednesday the 10th. We agreed to discuss the form during Wednesday's portal jam session and then put it into QuickPlace for everyone to access.

I spoke with seven developers that afternoon about what they liked and disliked about the portal jam sessions. Ben had yet to be assigned a task for the new portal so was not paying too much attention to the details of the jam sessions and did not feel any time pressure. Mike described his task to me in detail, it was a critical part of the new portal, but he said he had no action items from the portal jam sessions. He said the categories that Brian listed

out seemed to be adequate and that he felt some time pressure. Joe said they had spent a lot of time on the navigation topic so far and he needed to talk to the team about his task. He expressed a strong desire to talk to the team early next week otherwise he would feel significant time pressure. He also said the portal jam sessions were good, that Brian did a good job of running them, and that he liked that there was lots of discussion. Ethan said it was good that names were put to tasks at the jam sessions, although it was clear to him what his task was beforehand. He said there was not a lot of time pressure but that he did not know of any set deadlines. Ethan said a risk was that the team was not defining expectations about functionality and was not matching these expectations up with decision making. Bernard liked the organization of the meeting, that he could see who was doing what and how projects were related. He said he felt no time pressure. Seema re-iterated her thoughts from earlier that day and Greg said he thought the categories at the last meeting were really high level, that if they were his tasks he would be worried that he wasn't sure how to go ahead with them. However, none of them were his tasks.

There was so much variation in people's opinions about the portal jam sessions that I was unsure if the sessions needed change. The decision form and Joe's desire to discuss his task seemed like a good beginning. I informed Brian of Joe's desired agenda topic and began a task that became significantly more challenging than it should have been: getting access to the QuickPlace tool.

Monday, May 8th, and my second portal jam session came quickly. Brian chaired the meeting, first reminding everyone to access QuickPlace, and to let him know if you could not get access. He also said that by Wednesday the team members should think about what decisions need to be made for their tasks, the pros and cons of each decision and why a recommendation was made. Seema said that sometimes it is just what you get told to do and Brian agreed. Then Brian said Joe had a topic he wanted to discuss, and Joe took over the conversation. A large discussion ensued that involved everyone at the meeting: Clayton, Brian, Ethan, Joe, Mike, Bernard, Seema, Greg and Frank, a developer who had just come back from vacation. They talked about the way a similar problem had been done in the past, the departments at E-Solutions Inc. that would have an opinion on the problem (e.g. Human

Resources), specific scenarios to understand the problem, the larger impact of the decision, possible solutions, even definitions of terms. It was great. Partway through the discussion Pat came into the room and answered some questions the developers had. This became a habit of his – in part to let the developers discuss without "the boss" around, and in part because he was often double booked with meetings (as was Brian) and attended whatever was possible. *I was still trying to understand how things operated. And while the discussion was great, for me the contents of the meeting were still pretty vague.*

Tuesday May 9th I spent at school working on the decision form. I sent it to Brian by about 3 or 4 pm. In my email to Brian I said that I would need time to introduce it and explain it to the group. I suggested that our agenda for the next day's portal jam session be partially Joe finishing the discussion for this task and partially me introducing the form. The form is provided in Figure 9.1.

Wednesday was challenging. I still did not have access to QuickPlace – ok, a technical detail, literally. I still did not understand everything in the meetings – ok, it would take time and at least Joe resolved his task at the beginning of Wednesday's meeting. I still was unclear on my role versus Brian's role – ok I could just "follow his lead". But Wednesday's portal jam session, was challenging because the decision form was not introduced to the team at all. I suppose I should have said something during the meeting, but I didn't. The truth is, a decision form was not the "solution" I would have employed. So instead of speaking up about the form during the portal jam session, I let the session run its course, and found it to be not as successful as Monday's meeting. In my opinion, the meeting, and thus the conversation, lacked a focused topic.

After the meeting I met with Pat and Brian saying I was a bit concerned because it just seemed there were no tangible tasks discussed at today's portal jam session. For example the breakdown of tasks hadn't been specified any more seriously than the previous week. I brought out the decision form to show it to them and referred to it. The outcome of that meeting was that I would chair the coming portal jam sessions. There was no more talk about the decision form.

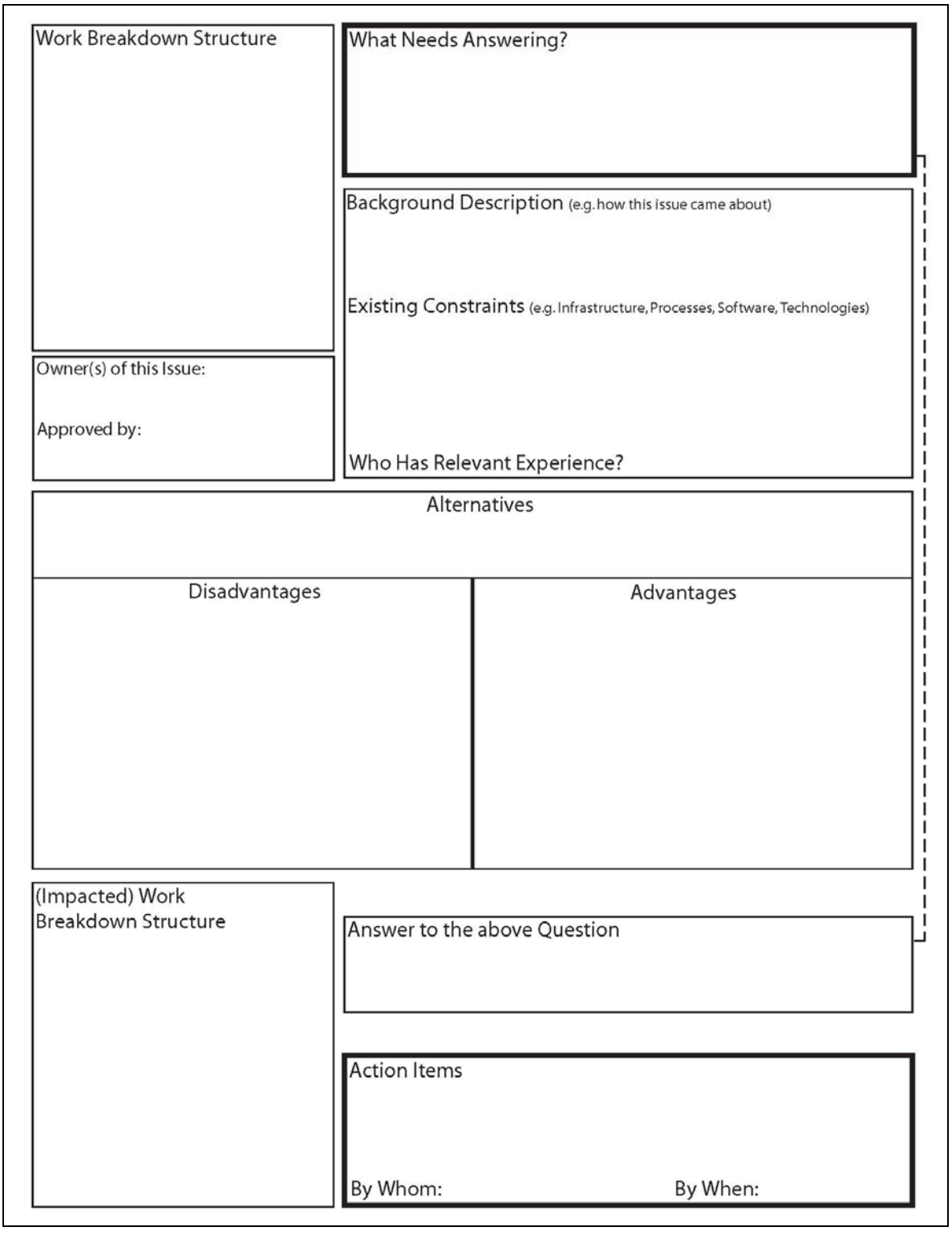

Figure 9.1: Initial Decision Form

The rest of the day I spent trying to meet with developers to discuss the format of the form. I introduced it to Seema, Clayton and Frank all at the same time and they had some questions but were ultimately open to trying it out. They didn't like the taxonomy section and we changed the wording on that. I introduced the form to Joe and Jeffery who helped me with some wording and also voiced concern over the taxonomy section. In the meantime I could not get access to QuickPlace so I could not post the form there. And when I finally did get access it was with limited privileges so I couldn't do everything I wanted to do.

The first portal jam session with me as chair occurred on Friday May 12th. I had asked Seema and Clayton to discuss the topic they were working on together. They used the decision form beforehand and presented their topic during the portal jam session. Similarly, Ethan and Frank discussed a topic they were working on together, during the meeting, although they didn't use the decision form very much before the portal jam session. The portal jam session was more structured but had less conversation than the meeting where Joe addressed his topic. There were small presentations by members of the team and they articulated the next steps of their tasks, but the members of the team who were not participating spoke up less than the previous meeting.

The second portal jam session with me as chair was "alright". Again, there was more structure in that there was a small presentation, but there was less discussion until partway through the meeting. Most disappointing was that no set action item came out of it. *Ethan presented his topic again and basically had to wait to talk to some people to get more information. Pat came in and answered questions a bit from a customer perspective. No one had a set action item when we left the session.*

During these days the different portal topics in which decisions needed to be made were becoming more clear to me. One task was what components of personalization would be included in the first launch of the new portal. Personalization was the portal content provided to a specific user, determined by his/her role in the portal. Customization, often discussed along with personalization, was allowing the user to modify the layout and the

look and feel of a portal page. Just to confuse the issue, the word security came up, which was a general term for that which accomplishes personalization. Existing user roles (e.g Executive, Employee, Contractor) were compared to existing company-defined roles (E.g. Strategy Shaper, Practitioner, Lead Practitioner) and the usefulness of these roles was discussed. Another concern was the level of abstraction at which personalization could be (technically) and should be (conceptually) defined. For example, would it occur at the portlet level (where a user can see certain portlets and not other portlets), or would it be defined at the page level, where a user can or cannot see an entire portal page (n.b. a portal page contains many portlets)? A component of this issue was the capabilities provided by Websphere – the development environment used to develop the portlets [Websphere07]. Another issue was how to incorporate world locations (i.e Canada, United States, South America, United Kingdom) into the idea of personalization. For example, what aspects of the United States portal can a Canadian user see? At the time, LDAP was used to control this. The goals of the entire company and how they intended to manage groups that were already defined seemed unclear: at one point Pat and Brian did not have the same information to give the developers discussing the topic. *In the end it was decided that "personalization as it pertains to Websphere will not be implemented until the functionality of adding/removing portlets is rolled out to portal users."* That is, the existing user groups defined in Canada (in part determined by the Human Resources department) would be enforced in the new portal, and the provision of individual personalization (provided in part by Websphere) would be pushed to a later date. All of this became only relevant to the Canadian portal as the launch of the new portal for the other three locations would wait until a later date.

A second issue that was discussed in tandem with the issue of personalization on a country-wide level, was the issue of the overall portal architecture. During some portal jam session I heard the question, "are we federating the content?" and had no idea what it meant. Three options were discussed at great length for how content should be organized in the worldwide portal architecture at E-Solutions Inc. The three options were Federated, Distributed or Centralized.

Federated: means each location (Canada, United States, South America, United Kingdom) will have their own portal (physical infrastructure and running portal instance). They can access a central location (e.g. in Canada) but they manage their own portal instance.
Distributed: seems like a more loosely defined version of federated, in my opinion. Each location would have its own box but it comes to a central location to get information (as copies). Requires synchronization with the central location. Difference between federated and distributed was blurry in what I read, but seems to be conceptual more than physical. Federated setup means each portal can produce their own information and consume it too (e.g. each location takes care of its own information and can access information from other places). Distributed is a copy only, the locations can't mange their own information and would go to a central location (e.g in Canada) for information from other locations (e.g United States, United Kingdom, South America).
Centralized: everyone comes to a central location to get information.
We met with experts on the topic, colleagues of one of the developers who had experience in this area, and they provided us with their thoughts on the issue. The positives and negatives of each approach – generic ones and ones specific to the company – were reviewed in detail and a recommendation to pursue the federated architecture was made. Interestingly, after this study was over I heard through the grapevine that an opinion from management might mean that a federated architecture was not executed.

The discussion of search was also part of the discussions about personalization and the portal architecture. Which search "tool" to use was linked to how much information had to be searchable for the first launch. Only one search tool was really pursued. The developer in charge of this task met with an expert on the tool to determine the tool's search capabilities and at the same time the developer was always asking for management's "ok" that the search only had to crawl Canada, for the initial launch of the portal. That is, a search that occurred in Canada only had to extend to the Canadian portal and the external website of the company, which was housed in Canada. Once this "ok" came from upper management, the developer recommended the tool he had been researching.

In addition, the choice of web content management had to be decided. “A web content management system is a computer software system that serves two primary functions: Facilitate collaboration among those responsible for maintaining content, … [and] Organize content.” [Flax06]. The existing web content management tool at E-Solutions Inc. had some disadvantages, a different web content management tool was pursued as a solution, and the advantages and disadvantages were provided. Related to this issue, and especially to the portal architecture decision, was how authoring would occur [Flax06] and how to conduct the migration to the new web content management tool. It was recommended to pursue the new web content management solution, rather than to stay with the old tool.

An unrelated topic was how navigation would look on the new portal. As I first started at E-Solutions Inc., fly-out menus were being implemented at, as I understood, the request of a member of management, who liked the look of it. While features of the fly-out were tweaked over time, alternative layouts were not considered, that I observed. The developer working on the task said that it was clear what was wanted and alternatives were not discussed because it was a specific problem.

These technical discussions are a sample of all the topics that arose during the portal jam sessions. Discussions on failover, clustering, and generating statistics of portal usage also occurred with or without me present.

Wednesday May 17th we had the third portal jam session, with me as chair, to discuss dependencies between the different tasks. It became clear that we didn’t have enough information to list out the dependencies. Greg came up with an idea to do an inventory of all the data/portlets in the existing portal, and where they were located. By the end of this portal jam session we had produced no dependencies between tasks. This seemed strange to me. One-on-one the developers would explain anything to me, and they were very generous with their time. Already Joe, Mike and Ethan had explained their tasks to help me understand and they understood roughly how their task fit with other tasks. That is, they had ideas about dependencies. After that meeting I thought about how I could change things. I also volunteered to take on the data inventory tasks, so I could learn more.

My idea for a change to the meetings was fueled by three things: Pat's original idea that all team members should be involved in the decision making process, my understanding of explanation based decision making and my observations that people were not using the decision form or QuickPlace to any useful measure. My intent was to motivate the documentation process and to have everyone glean enough knowledge to participate in decision making on all tasks. One caveat I knew about was that time was not on my side. We really only had about 10 business days until the end of May, when all the decisions were supposed to be made. I was unsure if this was enough time to bring everyone up to speed technically **and** make all the decisions that needed to be made. One caveat I did not know about at the time, nor did I have experience with as a contractor, was the intricacies of change management. Nevertheless, I moved forward and put together the idea presented in Figure 9.2.

The Potential Introduction of Knowledge Fest Meetings
a.k.a Changing the Portal Jam Sessions
Carmen Zannier

May 18, 2006

Purpose: For intense discussion about architectural and design decisions related to a software project.

General Concept: To introduce the idea of a game (or a small competition) about who knows more about their tasks, within a development team. A list of known and unknown items exist in some group knowledge base, as well as a status/report of winners of each meeting.

Roles:
Coach
The coach should be someone who is strong in the technical aspects of the project, has a high level view of the project and priorities, and is in a position of authority (in case it is needed to put pressure on high priority items). The coach should not be part of the development team. During the meeting the coach remains quiet during the meetings unless the conversation is not resulting in action items or a conversion of unknowns to knowns.

Knowledge Team
All members of the development team who participate in the knowledge fest (i.e. portal jam session) meetings.

Chair
The person who is in the "hot seat" for a meeting. S/he presents the items s/he doesn't know about during the meeting and the objective is to learn more during the meeting and focus action items.

The Process:
The Day before Each Meeting
Each member of the knowledge team posts a list of what they know about their individual projects (on QuickPlace). This can occur at an abstract level, but the more detailed, the better.

Each member of the knowledge team posts a list of what they *don't* know about their individual projects (on QuickPlace). This can occur at an abstract level, but the more detailed, the better.

In the course of listing what they know and don't know, they also list decisions they have made regarding their aspect of the project.

These posts have to occur at the same time each day (e.g noon), earlier in the day, so as to give structure to the process and to leave time for the coach to perform his/her task. If someone is unable to post their knowledge items by this time, they must discuss it with the coach first.

By the end of the same day the coach assigns a chair of the meeting based on the knowledge team's posts and who knows the *least* about their project. The person who knows the least is the person with the smallest ratio between their know:don't know lists. Because the coach is technically knowledgeable and has a high level view of the project and its priorities, s/he can assign subjective values to each developer's knowledge. The end goal is to pick the person who knows the least about their topic which has the highest priority, relative to other unknowns on the project.

The Day of the Meeting
The chair has some "cost" to being assigned chair (e.g. has to bring in coffee for the team).

At the meeting the chair should be set up with 3 boards on which s/he has written the following:

- Board 1: Titled "Knowns" with a list of all the project-related items that s/he knows.
- Board 2: Titled "Unknowns" with a list of all the project-related items that s/he knows and potential timeframes in which these items will be discovered.
- Board 3: Titled "Action Items" which is initially blank and by the end of the meeting will list specific tasks s/he has do by the next meeting.

The objective of the meeting is for the chair to get as much information as possible in the allotted hour with respect to the items listed on board 2 (Unknowns). Item by item these points should be addressed and either moved onto board 1 (Knowns) or translated into

action items.

Given this meeting the person that chaired the meeting should be in a better position to post a better known:unknown list, before the next meeting. Thus, the person who is the chair of the meeting, should rotate throughout the team as it is discovered who knows least about their task.

<u>After the Meeting</u>
The chair updates the knowledge list and action items (e.g. in QuickPlace) given the outcome of the meeting. In theory, if there was no outcome, s/he should "lose" again at the next meeting.

Nb: A continuous appearance of the same person chairing the meetings is a sign of trouble: perhaps the person has too much work, perhaps the conversations during the meeting haven't resulted in specific action items.

<u>Additional Considerations for Acceptance</u>

- Depending on the team a time box can be set up for the topics listed in the chair's unknowns items. This would be clocked by the coach (or the team with a super cool 10 minute "hour glass" ☺). For example, if the meeting is an hour long, and there are 5 unknown items, each topic gets 10 minutes of conversation (leaving 10 minutes for administrative or wrap up). After 10 minutes the unknown item must be converted to an action item if an answer hasn't been provided.

- The chair has some group-defined "cost" to ending up as chair, for example s/he has to bring in coffee for the group.

- The person with the most decisions made is rewarded by some group-defined award (e.g. team mascot).

- It's really important that these 3 "boards" (whiteboard columns, standup paper boards) exist so that each item is addressed one by one.

Recommendation for Introduction

- A meeting to explain the format of the meeting and generate ideas of a "reward" and a "cost" (something minor and fun, but meaningful).
- Enforcing posting what they know and don't know (specifics!) by tomorrow at … time?
- Trial period next week ?… need to decide timeline.

Figure 9.2 Proposed Changes to Portal Jam Sessions.

Thursday May 18th – We had a team meeting (not a portal jam session) *at which I expressed some struggles with the meetings and how to facilitate them. … [Later that day]*

an email was sent from one of the developers to Pat, Brian and me, communicating two things: input for the developer's development task and that the developer wasn't entirely happy with the portal jam sessions. That they were basically not getting anything done:

> *" The second thing is not my input to the JAM, but about the Jam. This may relate to what Carmen mentioned this morning, and is already being addressed, but I'll throw it out there anyway. The Jam sessions, which the exception of slide-out navigation, and a few other pieces, are still discussions of what could be done, not what can be done, should be done, or will be done. And as Brian has mentioned, we are getting very close to go / no-go decision time. Most of the discussion lately seems to be something that should occur off line from the meeting, in other scheduled or impromptu meetings, initiated by the person(s) responsible for the technology area, with input and/or attendance by anyone who has any knowledge and/or contacts for knowledge in that area. Then the results are delivered at the Jam, even if the message is, "we are still looking into that, and expect an answer by..."." –email.*

That was challenging, really, because I was about 1 minute (literally) away from meeting with Brian to propose my idea for a change to the meetings. It was also disappointing because the developer who sent the email could have posted the development task information to QuickPlace I proposed the design fest idea to Brian, and Pat jumped into the meeting. They both seemed open to it, asked some questions and allowed me to present the next day.

Friday May 19 – I presented my ideas and there wasn't exactly resounding applause. People were open to listening, had some questions, but weren't totally on board with everything. They said they'd like to see an example of how the meeting would go. So we decided that I'd be the first one to go into the hot seat on Wednesday, on a topic about which I know nothing. Pat told them they can see how it goes with the example. Then he said the presentation was good, …. He said they'll go through with it, and ultimately leave

it to the group to decide whether or not to continue trying it out. He said "it will be interesting to see what happens."

Did they even want me to do something this drastic? Yet at one point Pat said it isn't that big a change, …. I was disappointed with the way the decision form went and yet now people seem open to this idea. I had a lot of troubles with QuickPlace which made implementing the form difficult too and that was just plain annoying…. What if the meetings would have been okay without any change? Yet people were grumbling about the meetings. And there was this idea of "end of May deadlines, end of May deadlines". I guess we'll see how it goes on Wednesday. I have to learn the WCM architecture at least a bit before Wednesday's meeting.

Tuesday May 23, 2006 – I prepared for the meeting tomorrow, updating myself on everything to do with web content management. People are joking around about having to post in QuickPlace but are doing so in a fun way (mostly, I think). On Friday Mike asked me a question about what to type into QuickPlace – like how detailed it had to be and I said whatever felt natural. He posted information today and sent an email to me and Brian. Later he said that it was quite natural to use and useful too because it helped him list what he didn't know. Ethan came over this morning and asked if that was what I wanted (what he posted). I asked him how he found using it and he said it was simple and easy because you could just "dump" information in there. I asked Clayton and he said it was easy to use. I used it myself for reporting and found the format simple to use, I could structure it however I wanted. It took a while for me to type everything in (~90mins) but that had nothing to do with the form or even with the tool (aside from once when I lost a small piece of information). It was just that there was a lot to type. Tomorrow we go through my points one at a time. I also attended an architecture meeting today about web content management and it helped to clear up some things for me. Ethan has been extremely helpful in bringing me on board and helping me understand. Brian today said that the meeting probably helped me hear architecture information, which was nice.

Wednesday May 24, 2006. Tough day today. I had the portal jam session where I took the role of chair, according to the new explanation of the meeting. I was talking about web

content management architecture. I wrote the headline question on the board "Are we going with Federated or Centralized architecture". Then I had 3 points that I wanted answered. Two of them were minor and got answered quickly. The 3rd was about clustering and led to a conversation about collaboration. The conversation was good and when it started to veer off topic I pulled it back in. By the end of the meeting we had 2 action items from the third point and the first 2 points were resolved. Pat came into the meeting and asked what we thought of the new structure. People were positive, said they liked the new structure. But then one of the developers said he didn't think the person chairing the meeting should be the person who had the most questions about things, it should be the person who owns the decision (which is Brian). So we decided to tweak the meeting format, and on Friday Brian is supposed to chair the meeting and is supposed to present the Federated/Centralized topic and his decision. One of the developers challenged whether or not Brian got anything out of the meetings that we have been having and Brian said he had but didn't say anything specific. Mike said he liked having to write out what you didn't know. Pat asked the group if everyone was prepared to do the work I had done for each meeting (equivalent of 1.5 hrs prep and 30mins follow-up) and people were quiet. One of the developers challenged the idea of having the person who has questions chair the meeting. Everyone liked the structure of the meeting though. So we will see how things go on Friday.

The developer who had the most concerns during the meeting came up to me afterwards to see if I was upset about what he had to say and I said no. We got into a big conversation that was actually really interesting. The developer basically said he came from an idea where E-Solutions Inc. hires consultants with expertise in different fields. They do their job. The person who owns the decision has to get information from those people in order to make the decision. He said his time was valuable and bringing everyone in the group up to the same level of knowledge isn't aligned with the fact that they're each hired for their own specific expertise. And as consultants they can't make decisions on things, they can just recommend. He also said that they should have been able to make these large architectural decision up-front, that he, Brian and Pat, have years of experience and should have been able to make these decisions upfront.

The developer made really valid points, but I disagreed with some things. First, Pat mentioned to me that he wanted participation during the meetings, he wanted people to have input in the decision making. It's pretty difficult to have this occur if all the meetings become are a status report of information. Each person brings to the table their decisions and where they're at with it and everyone listens to see how it impacts their own decisions. I think the entire culture of such a meeting is different than a participatory meeting where people have a say in what happens. Also, they want these decisions to be tracked in QuickPlace. I think people's motivation for tracking decisions and sharing knowledge is going to diminish as soon as you take away the culture of participation. You change it to a "here is my work, here is your work" and they have less of a reason to share knowledge. Similarly, people's motivation to attend the meeting diminishes when they're just listening to someone tell them a status report, rather than actually having a role in the meeting. In addition, there wasn't ***a lot*** *of time pressure here, there was room for communication to occur rather than edict from above.*

A truly valid point is that time has gone by and it's difficult for them to continue with the meetings as they were, no matter what, because they're reaching the end of their time-box. Pat has said that they have already made a lot of decisions and now the structure of the meetings have to change to make some final decisions. I wish I had had this idea back when they had time to experiment because I feel like we cannot fully explore the challenges in part because we need to make decisions ***now*** *(so does that mean my idea is more of an information gathering thing rather than forcing a decision?). So now I need to think of a decision process that recognizes time as well as the culture's value system/management's desire for the outcome of the meetings.*

On Friday May 26 we had our portal jam session led by Brian. During this portal jam session Brian presented the 5 different ways that the architecture gurus suggested to do the portal architecture. There was discussion, primarily among Brian, Greg and Ethan. Seema looked confused at one point, Joe, Mike and Jeffery were mostly quiet except for a few questions. The entire structure of the meeting was different than the last few. Brian

presented the work and at the end of the meeting said, "That's all I have for you guys today". So it was quite top-down. But I can't say that it wasn't productive. We discussed how Brian had made a decision and he presented this decision and then they listed the other decisions they had to make that came out of that. So in that sense it was a productive meeting, it was just a completely different setup than the other meetings – no attempt at participatory decision making. At the end of the meeting Brian said, "So I'll put this into QuickPlace. Unless Carmen captured it all." I was concerned because people aren't consistently using QuickPlace, and I don't think this style of meeting lends itself to continuously using QuickPlace – there is no motivation for the decision maker presenting his/her decisions to put information into QuickPlace when it's duplicate communication. It's just a documentation process.

At times I was confused about the role of "decision facilitator". Sometimes it was as though all that was desired *was a person to take notes at the meetings and follow up with people about their tasks…. This suggests how much developers do not put in* ***extra*** *effort for decision making, they just collect information and then do something. The consistent "good" or "successful" thing has been continuous information gathering about topics. The consistent "bad" or "unsuccessful" thing has been imposing a format or set approach to decision making – decision form failed, QuickPlace is in the middle of failing, significant change to meeting format failed. Letting time go by for the developers to get the information they need has been consistent.*

9.2 Optimal Design Solutions?

Again, the first conclusion that emerged is that software designers appropriate design solutions approximately as often as they strive for an optimal design solution when making a design change. I will show this through reference to the case study described in Section 9.1 and through potential alignments with Explanation Based Decision Making [Penning93].

9.2.1 Auto-ethnography

There were two points I participated in that are evidence for the idea that software designers appropriate design solutions approximately as often as they strive for an optimal design. First, the scope of the project at E-Solutions Inc. was big, and there were so many components of it, that there just was no optimal solution. The project was more about trade-offs. If we "optimized" one component in one respect, it dictated a potentially sub-optimal solution of another component. For example, if the portal architecture centralized all content it meant the authoring aspect of the web content management (which was not technically detailed in Section 9.1) was ideally supported. One authoring "box" would exist at the central point. If the portal architecture was federated then the ideal authoring setup would be to federate the authoring boxes as well. That is, each location would have its own authoring box. However, this meant providing administration and resources for web content management at each location, a sub-optimal solution financially. One might say we just needed to pick the way in which we were going to optimize the solution, the point of view for optimizing. However, there were so many points of view involved in the discussion that this was difficult to resolve. It was the developer's job to recommend optimality from a technical point of view, the job of technical support to recommend optimality from a resource point of view and (perhaps) the project manager's job to recommend financial optimality. These were all trade-offs, depending on which way I looked at the problem. That is, we did not define an optimal solution or boundedly optimal solution for all dimensions.

Second, there were some decisions where alternatives just were not discussed. I can think of two such decisions: the navigation task and the approach for migrating to the new web content management solution. In the navigation task I was not present for the entire scope of the work, but in speaking with the developer and learning about the problem I found out that the idea of fly-out menus was prototyped and when it worked that is what was used. For the tool used for automated migration to the new content management solution, the developer involved learned of the tool from a colleague, researched the tool to determine if it would meet the needs of the migration, and ultimately recommended the tool. Given that these decisions were not compared against alternative solutions it wasn't clear if they were

optimal, or why they would be optimal, it was only clear that they satisfied the need of the team at the time.

From these two points I formulate my first observation of this study.

OBS1: There was no optimal solution defined for our larger design problem and for at least some of the smaller design problems.

This observation is evidence for the idea that software designers appropriate design solutions approximately as often as they strive for an optimal or boundedly optimal design solution because it is extremely difficult (arguably impossible) to strive for an optimal or boundedly optimal design solution that does not exist or is not defined.

9.2.2 Explanation Based Decision Making

One point from explanation based decision making leads to an optimal decision solution, however facets of the team I participated in were not aligned with this point. The point is that reasoning about the evidence is a central process in decision making [Penning93]. From my experience with the portal team members of E-Solutions Inc., they would not explicitly disagree with this, nor would I. However, on a practical level, always participating in reasoning about the evidence was difficult to do. Members of the team said the portal jam sessions were time consuming, a waste of their time because their tasks were not being discussed, not part of their contract (and no allowance was made for other work that was part of their contract), or not of merit because the person who owned the decisions should make the decisions (n.b the senior designer owned almost all the decisions discussed in the portal jam sessions). The developers often did not see personal value in their participation and were thus reluctant to participate. Given that their participation in the portal jam sessions was a critical opportunity to reason about the evidence, I can say I observed inconsistent support for reasoning about the evidence. I state this as my second observation:

OBS2: Designers inconsistently supported established techniques such as the portal jam session, to reason about the evidence.

Given that reasoning about the evidence is critical in explanation based decision making to achieving an “optimal” story, the results are evidence for the point that software designers appropriate design solutions approximately as often as they strive for an optimal or boundedly optimal design solution.

9.2.3 Summary of First Conclusion in Third Empirical Study

There were two points contributing to the idea that software designers appropriate design solutions approximately as often as they strive for an optimal or boundedly optimal design solution. The first is that there was no optimal or boundedly optimal design solution that was defined for many of the decisions we had to make. The second was that designers inconsistently supported reasoning about the evidence, a critical point of achieving some form of optimality in explanation based decision making.

9.3 Approach to Design Decision

The second conclusion that emerged is that when making a design change, the approach a software designer uses is impacted by his/her understanding of the design problem. I used the strength of a mental model as a definition of understanding, thereby recognizing the cognitive aspect of design. To show that a designer’s understanding of the design problem impacts the approach to decision making I will show that stronger mental models lead to serial evaluation [Klein98] and more open mental models lead to consequential choice [Luce58]. The evidence for this emerged in two ways: through reference to the case study presented in Section 9.1 and through a lens on Explanation Based Decision Making.

9.3.1 Auto-ethnography

The majority of the decisions we had to make required significant understanding of a larger architectural picture. There were numerous dependencies between a decision and the implications of a decision on other decisions that were constantly being discussed. In addition, the different perspectives on a decision brought different questions and answers to the problem. The discussion that occurred on Joe’s task is a prime example of this. No one involved in the discussion understood all aspects of the entire project. Brian arguably understood the most about the decisions discussed but there were some details that even he

learned in working with other members of the team (e.g. details about the benefits of a federated architecture and organizational information impacting the problem, the latter of which came from Pat). That is, there was no strong mental model shared among the members of the team, and individual mental models varied in strength depending on the person and the task.

In addition, the majority of the decisions made employed consequential choice. The personalization task, the overall portal architecture and its impact on web content management are all examples of the use of consequential choice. Here the developers did not know enough about the task to make an immediate decision, so they discussed the advantages and disadvantages of the task with team members and colleagues outside of the company. From these discussions they made a decision. The navigation task is a prime example of a problem that was clearly defined to the developer and serial evaluation was employed. Similarly, in defining search, once the developer had the information desired about the capabilities of a search tool satisfying the search requirements of the company, he did not consider any more alternatives and no one challenged him on his choice. That is, he used serial evaluation. These examples are evidence for the conclusion that a developer's understanding of a design problem impacts the approach taken in decision making.

9.3.2 Explanation Based Decision Making

One point of explanation based decision making aligns with what occurred in the decision making and the idea that understanding impacts the approach to decision making. In explanation based decision making stories built as potential solutions to decision problems may contain stories inside themselves, so that the story is actually a hierarchy of stories [Penning93]. This aligns with the decisions I observed and participated in at E-Solutions Inc.. One developer would raise a decision topic for discussion which would start a conversation that expanded into numerous sub-topics or sub-stories. At first, this made the discussions difficult to follow, but as my knowledge of the problems improved I began to appreciate the dependencies between the problems and the potential hierarchy of stories that could be described. With this understanding the team could consider the impact of a decision on other decisions and consider the trade-offs involved – such as was mentioned in

Section 9.1. Without this understanding the team simply did not consider the impact of a decision. They just let the decision be made – such as in the selection of a search tool. The key point though, is that to understand the dependencies between the problems was to understand the lack of a "proper solution". In other words, understanding the dependencies made the problem more complex, and more difficult to develop a strong mental model, especially shared among members of the team. In this respect a strong mental model led to the use of serial evaluation and an open mental model led to the use of consequential choice.

There were six decision points highlighted in the thick description of Section 9.1, and these are shown in Table 9.1 These decision points show the strength of the mental model and the use of consequential choice or serial evaluation.

Table 9.1 Summary of Design Decisions at E-Solutions Inc.

Case Study ID	Description of Decision	Approach to Decision
CS3.1	Personalization in the first launch of the new portal.	Consequential Choice
CS3.2	Portal Architecture	Consequential Choice
CS3.3	Which portal search tool to use	Serial Evaluation
CS3.4	How much information had to be searchable for the first launch of the portal.	Serial Evaluation
CS3.5	The choice of a web content management tool.	Primarily Serial Evaluation
CS3.6	How navigation would look on the new portal.	Serial Evaluation

9.3.3 Summary of Second Conclusion in Third Empirical Study

There were two points contributing to the idea that software designers' understanding of a design problem impacts the approach they take to decision making. The first point was the complexity of the design problems at E-Solutions Inc. meant that no one had complete understanding of all the decision that needed to be made, which led to talking to other team members about alternatives. The second point was that the complexity of the design problem led to a hierarchy of stories, as per explanation based decision making, and the use of consequential choice.

9.4 Study-Specific Results

There are two components I'd like to address, that will overlap each other throughout the discussion of lessons learned. The first component is the set of results relative to decision making in general, as was done in the last two studies. The second component is the set of results relative to impacting decision making and implementing change in a team. The topics addressed below are: explanation based decision making, the use of time boxes, personal value and aligned goals in group meetings, and finally, change management.

9.4.1 Explanation Based Decision Making

My experience at E-Solutions Inc. and my understanding of explanation based decision making leads me to say that explanation based decision making is a good "start" for describing design decision making. I have four points to support this. The first point is that reasoning about the evidence, as per explanation based decision making, aligns with what occurred at E-Solutions Inc. The content of the portal jam sessions and the content of informal meetings I overheard while working were centered around technical ideas or questions developers had and generating opinions or answers from other developers on the team. Thus, "reasoning about the evidence" is an appropriate description. The second point is that the chosen decision option has to have a "goodness of fit" for the decision problem, as per explanation based decision making. Again, this was an appropriate description of the events at E-Solutions Inc.. Given the number of dependencies among decisions and the tradeoffs therein, the best we could do was make design decisions that "fit" with all the decisions we had previously made. It was more about structuring the problem into a reasonable solution than finding some optimal solution. However, the third point shows why explanation based decision making is only a "start" to describing design decision making. Explanation based decision making aligns decision alternatives with decision solutions and the best pair is selected. I cannot say that we paired information about the decision problem with solutions explicitly. Again, the emphasis was on tradeoffs of solutions. Implicitly perhaps this was done, but it was not a concrete aspect of the discussions in which I participated. Lastly, explanation based decision making was viewed as a "start" to software design decision making because of what it lacked. Direction on personal value, goal alignment of those involved in the decision making process, a suitable

number of people involved in reasoning about the evidence were all required, and not provided by explanation based decision making. For these four reasons, the first study-specific result from this empirical study is that explanation based decision making is at best a “start” to describing design decision making.

9.4.2 Time-boxes and Time Pressure

We had a time box of one month. As we neared the end of the time box and felt time pressure, Brian made the decisions on the topics we had discussed. If we had not spent time discussing the alternatives before this pressure was felt, I would argue we did not use consequential choice. As it was, our approach involved less and less consequential choice the more we approached the end of the time box. Thus, an observation point is that time pressure encourages the use of serial evaluation. Another observation point is that clearly some decisions are time-boxed (as opposed to the result from the second empirical study).

9.4.3 Personal Value and Aligned Goals

If a group of people meet in a room they should a) all understand why they are there and b) see tangible value in their participation. Perhaps this sounds trivial, but it was one of the largest hurdles of the portal jam sessions at E-Solutions Inc. One of the initial goals of the sessions was that all members of the team should participate in decision making, yet this was not well communicated to the developers, or if it was it was not well received. Or both. developers sometimes said the meetings were a waste of time because their tasks were not being discussed, their participation was not part of their job description, they were not compensated for their time in the meetings, and/or the end result of the meetings was not in their “control” (i.e. they did not own the decision). If I had to do it all over again, I would re-iterate – at nearly every meeting – why we were there and that the overarching goal was group participation in decision making. In addition, I would assist in defining personal value for each developer who attended the meetings, focusing on individual change management [Lewin51].

9.4.4 Change Management

The observation was that change management requires explicit, unwavering support at the senior levels of the organizational hierarchy. In my opinion, this did not occur at E-Solutions Inc. to the extent that change management was successful. In total, three changes were proposed – the use of a decision form by developers, the use of QuickPlace by developers to record information about decisions made, and the new structure of the portal jam sessions. During my experience tactical action was not strong enough for meaningful change to occur [Beckard69].

9.5 Chapter Summary

The results of this chapter are evidence for the two main conclusions and set of eight smaller results. The first main conclusion is that software designers appropriate design solutions approximately as often as they strive for an optimal or boundedly optimal design solution. The second main conclusion is that the understanding of design impacts the approach to decision making. Both these points were shown through the case study description provided in Section 9.1 and interpretations on it, and through a lens on explanation based decision making. The first smaller result was that explanation based decision making is a great start to describing software design decision making. The second result was that approaching the end of a time box encouraged the use of serial evaluation. The third result was that personal value and aligned goals should be made explicit when asking software developers to participate in design decision making. The fourth result was that change management requires explicit and unwavering support from senior members of the organizational hierarchy. These results are used as part of a cross case comparison with the first two empirical studies.

CHAPTER TEN: CROSS CASE COMPARISON RESULTING IN DEISGN DECISION MODELS

The three empirical studies presented in this thesis were similar in purpose and scope, but different in execution, to determine if different execution would lead to different results. From the first study to the third study the techniques used to collect, analyze and report data were increasingly qualitative. The first empirical study used a semi-structured interview format with 25 interview participants from the software development industry. It also used frequencies of codes and relationships between codes that emerged from interview transcripts and directed the provision of quotes and case study summaries. The second empirical study used on-looker observations of software developers performing software development. It also used pre-defined codes on case study summaries, and proposition building to contribute to the final results. The third empirical study used participatory observations of design decision making and an attempt at action research. It also used auto-ethnography to contribute to the final results. In my opinion, the differences in the execution of these studies satisfy the idea of triangulation because they provide different research methods to addressing the same research topic [Patton02]. This makes a comparison of these three studies valuable. I compare the results of each of the three multi-case studies by comparing final results provided in Chapters 5, 7 and 9. Throughout this chapter, I refer to specific cases within each multi-case study, as needed. Figures 10.1 and 10.2 show the argument structure across the cases. This chapter concludes by consolidating the three multi-case studies into 5 models of design decision making.

10.1 An Optimal Solution?

Across the three studies there were eight observations that were evidence for the conclusion that software designers appropriate design solutions approximately as often as they strive for an optimal or boundedly optimal design solution. Four of these points were results of the first empirical study, two were results of the second study and two were results of the third empirical study. In this conclusion I have used the phrase “approximately as often” because the number of case studies supporting ideas like Satisfice and Serial Evaluation are not recognizably greater than the number of case studies supporting ideas like Weight and

Consequential Choice. However, these numbers are close (e.g. 13 vs. 12, out of 25 case studies).

The first observation was that the code **Better vs. Worse** occurred more often than the code **Right vs. Wrong**, and the code **Satisficing** occurred at least as often as the code **Weight** when interviewing software designers about design decision making. For example, for interview participant #14 the code **Better vs. Worse** constituted 5.8% of all codes assigned to the transcript during small window coding and 9.1% of all codes assigned to the transcript during large window coding. The code **Right vs. Wrong** constituted 3.3% of all codes assigned to the transcript of interview participant #14 during small window coding and 2.3% of all codes assigned during large window coding. This led to the second observation, which was the provision of quotes exemplifying Designers discussing software design as **Better vs. Worse**, rather than **Right vs. Wrong**, when interviewed about software design. A prime example was the following quote,

"There is nothing like working with good examples, … working with good code, other people's good code…"

The first proposition of the second empirical study aligned with these two observations. This proposition stated that the representation of software design is predominantly framed in subjective terms. This was found by observing software designers making software design decisions. The fourth observation was a statement of experience from participating in software design decisions. This statement was that there was no optimal solution defined for a large design problem and for at least some of the smaller design problems that made up the larger design problem.

These four observations (found in Figure 10.1) show that at best, software designers can strive for a boundedly optimal design solution. A design solution is defined in subjective terms, which has bounds defined by an individual's interpretation of the subjective terms. All three studies are consistent with each other in showing a lack of quantifiable optimality in design solutions. There are four remaining points that are evidence for the rest of the conclusion.

The fifth observation is that the code **Satisfice** occurred at least as often as the code **Weight**, when interviewing Designers about software design decisions. That is, just over half (13/25) of the interview participants discussed satisficing more than discussing the use of weights on options in a decision. This led to the sixth observation which was the provision of quotes exemplifying Designers discussing a satisficing approach to design. A prime example was the quote,

“”… if [an option] is not obviously a bad thing then its probably good enough for my purposes for now.”

The second proposition in the second empirical study aligned with these two points. This proposition stated that designers frequently evaluate one alternative at a time instead of comparing alternatives in their design decision making. This was found by observing software designers making software design decisions. The final observation was a statement of experience from participating in software design decisions. The statement was that designers inconsistently supported reasoning about the evidence surrounding a design decision. That is, they were not always interested in considering alternatives to design problems.

These last four observations (found in Figure 10.1) show that designers used satisficing or serial evaluation, which by definition employs satisficing, in order to make a design decision. Thus they often did not consistently strive for a boundedly optimal design solution. The three studies are consistent with each other in showing a lack of pursuing a boundedly optimal design solution.

Given these eight observations and the consistency across them, I conclude that software designers appropriate design solutions approximately as often as they strive for boundedly optimal design solutions.

An immediate question of this point is what were the designers trying to optimize? From the data provided in chapters 5, 7 and 9, the designers tried to optimize “good” design, but “good” was never explicitly defined. At times it was following a design heuristic (e.g. separation of presentation and business logic), at times it was the improvement of workflow

in the software development team, and at times it was getting rid of a bug. Good, optimal, or boundedly optimal design was not consistently defined, and this could hypothetically be seen as part of the reason the designers appropriated so many design solutions.

<table>
<tr>
<td>DATA: Case 1
Frequencies of codes Better vs. Worse occurring more than Right vs. Wrong</td>
<td rowspan="4">ARGUE:
At best software designers make boundedly optimal design decisions, where the bounds are defined in subjective terms.</td>
<td rowspan="8">CONCLUSION 1:
Software Designers appropriate design solutions approximately as often as they strive for an optimal or boundedly optimal design solution.</td>
</tr>
<tr>
<td>DATA: Case 1
Emerging relationships</td>
</tr>
<tr>
<td>Proposition 1: Case 2
Representation of the value of software design is predominantly framed in subjective terms.</td>
</tr>
<tr>
<td>Observed Experience: Case 3
There was no optimal defined for our larger design problem and for at least some of the smaller design problems.</td>
</tr>
<tr>
<td>DATA: Case 1
Frequencies of codes Satisfice (13/25) & Weight (12/25), and 12/25 interview participants referencing a good enough or satisfactory design.</td>
<td rowspan="4">ARGUE
Designers who do not compare alternatives do not strive for (even) a boundedly optimal design solution.</td>
</tr>
<tr>
<td>DATA: Case 1
Responses to Questions</td>
</tr>
<tr>
<td>Proposition 2: Case 2
Designers frequently evaluate one alternative at a time (13/28) instead of comparing alternatives in their design decision making (15/28).</td>
</tr>
<tr>
<td>Observed Experience: Case 3
Designers inconsistently supported established techniques such as the portal jam session, to reason about the evidence.</td>
</tr>
</table>

Figure 10.1: Argument Structure of First Conclusion

10.2 Approach to Design Decision

Across the three studies there were eight observations that were evidence for the conclusion that a software designer's understanding of a design impacted his/her approach to decision making. Three observations were the result of the first empirical study, three observations were the result of the second empirical study and two observations were the result of the third empirical study. This argument structure is shown in Figure 10.2.

The first observation showed codes relating to software design and codes related to understanding as being somewhat prevalent in the top three codes of interview participants. For example, in large window coding 13 of 25 interview participants have a code from both Design and Understanding code categories, in their top three codes. This led to the second observation, the quotes qualitatively showing the relationship between Design codes and Understanding codes. A prime example of this is quote O5.30,

"…I always try to encourage people to think from other parts of the design… so one of the basics is that very isolation of the domain design decisions from other decisions."

The third proposition of the second empirical study aligned with these points. This proposition stated that designers use mental models and mental simulations while making design decisions. This was found by observing software designers making design decisions. The fourth observation was a statement of experience from participating in software design decisions. This statement was that the complexity of the design problem led to a hierarchy of design stories that required acknowledging tradeoffs in a design solution.

These four observations (found in Figure 10.2) show a relationship between ideas designers have and software design, specifically that designers have ideas and series of events (i.e. stories) about design, in their head, as they make design decisions. More eloquently put, designers employ mental models and mental simulations when making software design decisions. While this observation may not be novel, these three studies are consistent in supporting it and each other, and the observation acts as an initial contribution to the second main result of this research. Four observations remain to complete the contribution.

The fifth observation is the provision of 20 case studies showing the strength of a mental model impacted design decision making. For example, interview participant #3 immediately saw favourable aspects of a design solution and went with that decision without considering alternatives. This shows the development of a strong mental model leading to serial evaluation. The fourth proposition of the second empirical study stated that uncertainty in software design occurred in more forms than uncertainty in real options analysis. That is, an internal uncertainty, or understanding of the project had to be resolved in order to solve the problem. In light of this, case study summaries were provided showing how the strength of a mental model impacted the approach to design decision making. The eighth observation was a statement of experience from participating in software design decisions. The statement was that the complexity of the design problems meant no one had a complete understanding of all the decisions that needed to be made, which led to talking to other team members about alternatives.

These last four observations show that designers' ideas about design can vary in detail and thus impact the approach taken to design decision making. The three studies are consistent with each other in showing the use of consequential choice and serial evaluation, depending upon the strength of the mental model.

Given these eight points (shown in Figure 10.2) and the consistency across them I conclude that a software designer's understanding of a design solution impacts his/her approach to decision making.

10.3 Secondary Results

There was one secondary result that occurred in more than one experiment. The result was that time pressure imposed the use of serial evaluation. This occurred in the first study with interview participants. Three interview participants discussed time pressure in their design change to the extent that it caused them to use serial evaluation in making a decision. In the third experiment as we neared the end of a time box, decisions were just made, with little acknowledgement to the reasons they were made. This example is not as strong an example as in the first study, yet the feeling of nearing the end of a time box did change the "feel" of

the portal jam sessions, in my experience. However, the secondary result here is: time pressure imposed the use of serial evaluation in design decision making.

<table>
<tr>
<td>DATA: Case 1
Frequencies of codes related to Design and Understanding</td>
<td rowspan="4">ARGUE:
Designers employ mental models and mental simulation in their design decision making.</td>
<td rowspan="8">CONCLUSION 2:
Designers use serial evaluation when they have a strong mental model of a design problem and they use consequential choice when they have an open mental model of a design problem.</td>
</tr>
<tr>
<td>DATA: Case 1
Emerging relationships between codes related to Design and Understanding</td>
</tr>
<tr>
<td>Proposition 3: Case 2
Designers use mental models and mental simulations when making design decisions.</td>
</tr>
<tr>
<td>Observed Experience: Case 3
A hierarchy of stories in decision making (dependencies between decisions) meant understanding a lack of a “proper” solution and being open to tradeoffs.</td>
</tr>
<tr>
<td>DATA: Case 1
Responses to Questions showing relationships between designer’s understandings of a design and their approach to decisions.</td>
<td rowspan="4">ARGUE:
Extent of understanding (i.e. strength of a mental model) can vary.</td>
</tr>
<tr>
<td>Proposition 4: Case 2
Uncertainty in software design occurred in more forms than uncertainty in real options analysis.</td>
</tr>
<tr>
<td>DATA: Case 2
Case study summaries showing relationship between designers’ understanding of a design and their approach to decisions.</td>
</tr>
<tr>
<td>Observed Experience Case 3
The complexity of the design problems meant no one had a complete understanding of all the decisions that needed to be made, which led to talking to others.</td>
</tr>
</table>

Figure 10.2: Argument Structure of Second Conclusion

10.4 Models of Design Decision Making

I have developed five models of design decision making that capture the various decisions processes the data describes. Figures 10.3 to 10.7 show the models. These models are firstly, descriptions of design decision making, and secondly, pointers towards future work. These models can be used in future empirical studies that can support, refute or refine the models.

There were two factors that defined the decision scenarios: the number of people involved in the decision making and the size of the design problem. There were six decision activities: rationalizing, experimenting, searching, comparing, discussing and considering. The five decision models involved these activities are: Single Designer Small Design Problem, Single Designer Medium Design Problem, Two to Five Designers Small Design Problem, Two to Five Designers Medium Design Problem, Five to Ten Designers Large Design Problem. I did not observe sufficient data to address the remaining situations of Single Designer Large Design Problem, Two to Five Designers Large Design Problem, Five to Ten Designers Medium Design Problem, or Five to Ten Designers Small Design Problem.

10.4.1 Design Decision Categories

During this research I was extremely hesitant to separate design into categories such as code, class design, architectural design, or the like. The reason I was hesitant was because the scope of the design problem was perceived by the designers involved in the task, and varied depending on his/her perspective. For example, in the first empirical study, interview participant #3 found numerous cues that a design change needed to be made, one being that the existing code was verbose and awkward. Interview participant #8 was told about difficulties handling customer requirements in an efficient manner, for a portion of an existing system. Interview participant #10 had pressure from a marketing department to incorporate XML and XSLT into a design solution. All three of these developers employed similar solutions to their design changes – the use of XML and XSLT in an existing system. However, all three of these interview participants arrived at this solution in different ways. In contrast, interview participant #13 and interview participant #17 both recognized in their

source code that they were mixing business logic and presentation logic in web applications, which does not align with object oriented design principles [Bruegge04]. However, they handled the situations differently (interview participant #13 manually separated the logic in a serial evaluation fashion, interview participant #17 used an existing knowledge source to compare a new solution (velocity templates) to a solution he had used previously (taglibs). These examples show designers recognized problems in different ways but found similar solutions, and recognized similar problems but found different solutions. Thus, categorizing design situations was difficult. I have selected three somewhat generic categories, recognizing that they are not mutually exclusive. I strongly believe assigning a design situation to a category depends on a designer's expertise and the context of the design situation. Given this, I used categories of small, medium and large design problems, as defined in Section 3.2.

10.4.2 Design Decision Activities

There were six activities that emerged across the cases. They are Rationalizing, Experimenting, Comparing, Discussing, Searching and Considering. For each definition I also describe where the term came from in the studies.

Rationalizing: The designer develops predictions and small theories about how the system works or how the system should work. The designer supports these predictions with problem structures and with explanation building. The problem structures come from knowledge s/he has learned of the structure of the system, his/her previous experiences, and knowledge that s/he has researched. The explanation building comes from the designer's ability to present a logical and winning explanation to his/her peers and from the designer's ability to challenge his/herself on the explanation, in order to strengthen it. This term came from three observations during the study: first, the manner in which interview participants sometimes provided explanations for decisions they had made (i.e. in a nonchalant, shrug of the shoulder manner, as if the explanation was not too important except to answer the question; second, from mumblings I observed of software developers working alone on design in the second empirical study, and third, from my understanding of the decisions made during the third empirical study.

Discussing: Multiple designers working together exchange information about their respective mental models. When more than one designer is present, there is an unstated assumption that the designers have a similar understanding of a design situation. Discussing is the small conversation that occurs often right before it is discovered that the designers' mental models are not aligned on some aspect of the system. This term came primarily from the conversations I observed during the second study. Conversation topics were initiated by someone stating their assumptions about a system out loud.

Searching: Any pursuit of supporting knowledge for a design problem, whether it be on the internet, in textbooks, in a conversation with a colleague about his/her past experiences, or the like. When a designer is working alone, the extent to which this activity is conducted depends on the individual designer. When a designer is working with more than one person, at least some search occurs naturally in learning about the other person's ideas during natural conversation. This term came from any and all reference to knowledge sources – textbooks, the internet, articles and the like – and this occurred throughout all three studies.

Comparing: Evaluating the advantages and disadvantages of more than one design solution. When a designer is working alone, the extent to which this activity is conducted depends on the individual designer. When a designer is working with more than one person comparing occurs somewhat naturally in learning about the other person's ideas, but also depends on the group dynamics and the designers' tendency to challenge each other. This term came from the conversations I observed during the second empirical study, the conversations I participated in during the third empirical study, and the references to helpful colleagues, mentioned during the interviews.

Experimenting: Coding a proposed design solution (e.g. prototyping) and testing it. The testing can occur at varying levels of detail, but is often minimal, to determine if the proposed design solution works. The extent to which experimenting is done on multiple design solutions depends on the size of the design problem. This term came from designers talking about "trying out an idea" or "seeing what happened", and from observing people continuously immersed in the source code when working on a design problem, during the second empirical study.

Considering: A joining of the activities Searching, Comparing and Discussing, separated in its own activity here for clarity in discussing the diagrams.

10.4.3 Design Decision Scenarios

There were three concepts that emerged in describing the five decision scenarios. They were Appropriating, Reasoning, and Explaining. The definitions of these are as below, with a justification for the use of the term, followed by the description of the five scenarios. Diagrams of the five decision scenarios are found in the following pages.

Appropriating: The application of a design idea to a design change, where the idea accommodates the problem, regardless of varying levels of awareness of design alternatives that the software designer making the design change may have. The first part of this definition is taken from the definition of appropriate as per [MerriamWebster07] and the second part suggests a non-extreme use of satisficing. This occurs in small design changes. The emphasis is on the activities Rationalizing and Experimenting, with some use of Discussing when more than one developer is present. This term was chosen because it encompassed serial evaluation, but left room for the idea of mental modeling.

Reasoning: The comparison of alternatives to a design problem using easily accessible methods of comparison. Such methods are internet searches, design patterns that a designer has some recollection of, generic re-use of past experiences, and most often, conversation with colleagues. This occurs in medium design changes. In decision scenarios with a single designer, reasoning highly depends on the traits of the designers and his/her desire to “do a good job”. In decision scenarios with multiple designers reasoning emerges naturally through conversations among designers who need to align their mental models and who feel comfortable challenging each other’s ideas. This term was chosen because it encompassed consequential choice but also incorporated the concept of negotiations that occurred when more than one designer was present.

Explaining: The relaying of design information to members of a team who need that design information to accomplish their tasks. This occurs in large design changes with five to ten designers. The “Construct Explanation” process stage found in Explanation Based Decision Making [Penning93], (i.e. in Figure 2.1) is aligned with the Explaining scenario that I observed. However the “Construct Choice Set” of Explanation Based Decision

Making is not immediately aligned with the Explaining scenario that I observed. The primary difference is that I mean Explaining in the sense of one sub-team explaining concepts and design decisions to other members of a sub-team, whereas Explanation Based Decision Making shows how a single team develops a design decision. This term was chosen because of the large group interactions I participated in during the third empirical study, where we split into sub-groups and then explained concepts to other sub-groups.

The first of the five decision scenarios is one of a single designer conducting a small design change. The designer rationalizes a potential design solution with him/herself and there is no one present to argue his/her rationalization. Thus, more often than not, the designer will move immediately to implementing his/her design solution. Because the problem is small the designer is able to repeatedly develop rationalizations (or ideas) and implement and test them quickly in the experimenting activity. The designer can cycle through this "idea-try it out" combination very quickly, until the problem is resolved. In this way serial evaluation is employed. The design solution depends on variables such as the designer's development skill, the development environment, and the designer's experience and personality. The design solution does not depend on evaluating the trade-offs of design solutions. This appropriating scenario is impacted by variables such as time pressure, organizational culture, and external pressures. An example case of appropriating is case study 2.7A. The designer read source code that had a bug in it, until he had an idea for how to solve the bug. He implemented the idea and tested it and when it worked he was done. He did not challenge the solution or try to understand why he had not provided the fix in the source code before. He fixed the bug and moves to the next task he had. The design decision scenario called Appropriating, for a single designer working on a small design change, is found in Figure 10.3.

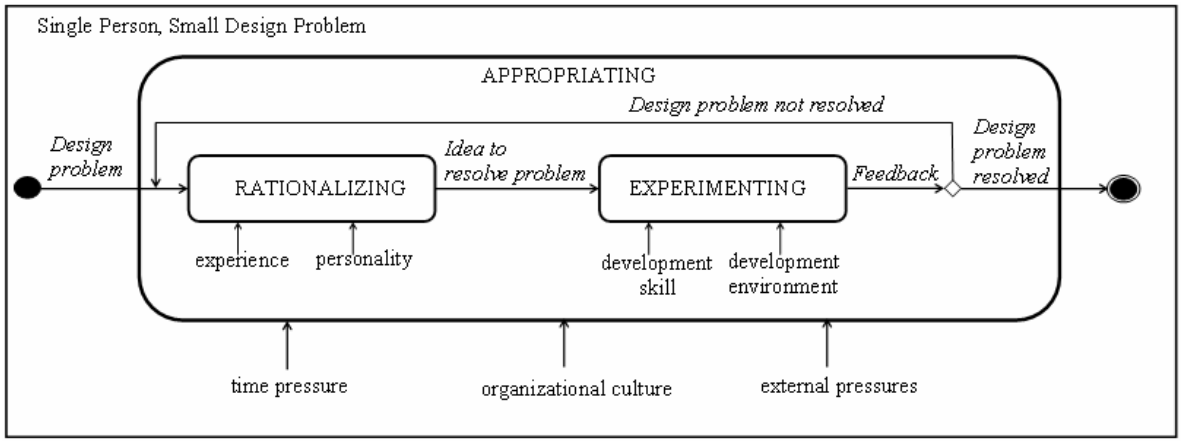

Figure 10.3: Single Designer, Small Problem Decision Making

The second of the five decision scenarios is of multiple designers working on a small design change. Similar to Appropriating in a single designer situation, the designers here rationalize a potential design solution with each other. For example, one designer proposes an idea (with some explanation as to why the idea is sufficient or even "good") and a second designer agrees. The two move to the experimenting activity to implement and test the idea. If the idea does not work the designers discuss aspects of the system that might cause a problem in implementing the design solution, and they make small changes to the system. In this way the designers cycle through this "idea-try it out" combination quickly. The primary difference between this Appropriating scenario and the Appropriating scenario with a single designer is that the multiple designers need to discuss their ideas with each other to align their mental models. Because the design problem is small, this occurs quickly, often with agreement between the designers. Disagreement suggests the design change is larger than first assumed. A prime example is case study 2.7B. Two designers, pair programming, both remember a way they solved a problem in a previous design situation, which they can use in their current design situation. They look for that source code, apply it to their problem, test it, and it works. The context variables are the same as in Appropriating with a single designer except that team dynamics also plays a role in how the designers work together. This Appropriating scenario is found in Figure 10.4.

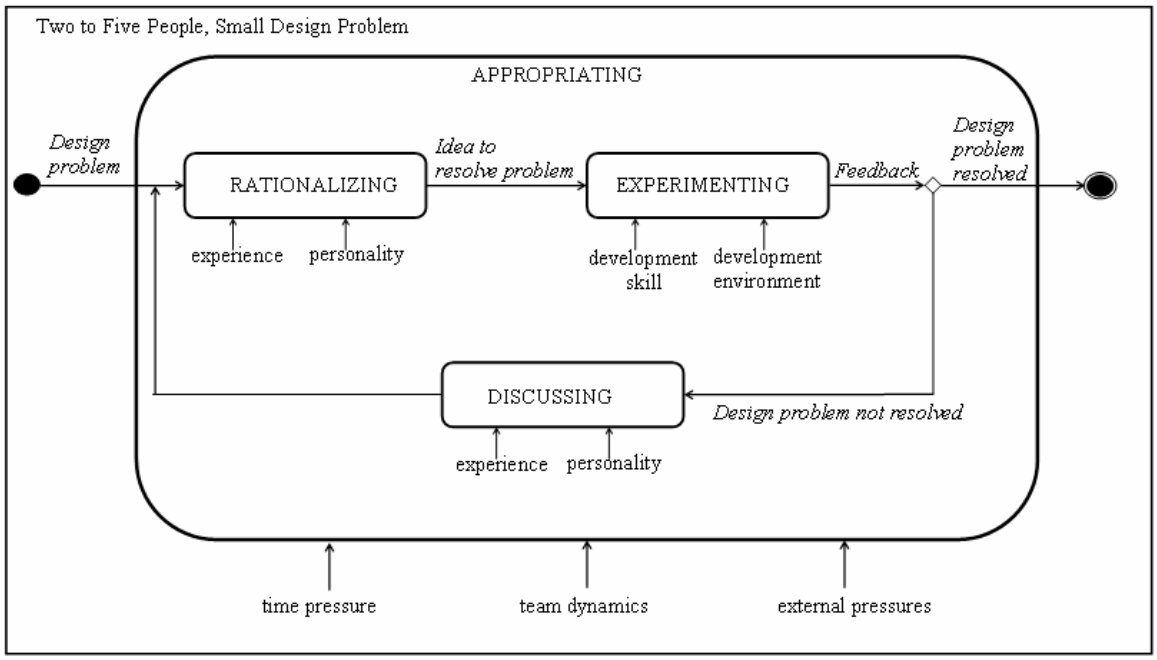

Figure 10.4: Two to Five Designers, Small Problem Decision Making

The third decision scenario is of a single designer addressing a medium design change. This employs Reasoning. A designer uses Rationalizing in that s/he develops ideas about how to solve a design problem, based largely on his/her past experience and ability to "sell" the idea to himself/herself. A designer uses Experimenting by prototyping a design solution which, when successful, support the rationalization that the idea was a good one. The difference between this Reasoning and Appropriating in a single designer scenario, is that the designer is aware that the design change is more significant than a small change. For example, introducing a new class and moving logic from one class to another is more significant than a quick bug fix. The designer is aware of this and its potential impact on other designers who are working on the system. Thus s/he will spend more time (than in Appropriating) in justifying the rationalization, through some searching and comparing of design alternatives. The searching may occur through conversation with colleagues who have relevant experience, or through quick internet searches, but it does occur. Similarly the comparing may be almost invisible and very quick, but it does occur. For example, in case study 1.4 the designer quickly recognized that he could implement a design solution that required an end user to take two steps in a workflow, or one, and quickly chose the approach that enforced only one step. The comparison activity is subject to traits of choice

found in the literature such as representativeness and availability [Kahnema82], and bolstering and deemphasizing [Montgom93].

In addition, the thoroughness of the search and compare depends on the designer, for Reasoning in a single designer scenario. This scenario had the most variation in the degree and quality of activities that were dependent upon factors not examined in this study. More specifically, the extent to which a detailed search occurred, and the extent to which multiple options (e.g. 2 versus 5) were considered when a developer was working alone was partially affected by factors such as personality and organizational culture. A concrete example is found by comparing CS2.1B to CS2.1C. When observing these two individuals, I was impressed by the detail and care that the observed designer brought to his design problem, in CS2.1B. The design problem seemed potentially trivial to me – part of the problem was a user interface issue – yet the designer spent a significant amount of time considering options, and addressed the user interface with a degree of care that I found impressive. In comparison, I did not see these same qualities in CS2.1C, yet the scope of the design problem was similar to the scope of the design problem in CS2.1B, in my understanding. In case there is a bias here, I can argue the exact same point when comparing interview participant #25 to interview participant #13. Both participants discussed a small pet project. Interview participant #25 discussed a desire to "kick the tires" on the project and did employ consequential choice in his design change. Interview participant #13 did not express such a desire, and did not consider alternatives in his pet project, despite the fact that his design change was of large enough scope to have some design alternatives (i.e. the solutions considered by interview participant #17, in the same problem domain). Factors such as personality and organizational culture were not explicitly pursued in this research, yet I developed a sense of these factors affecting the decision making process, and this became apparent in the decision scenario with a single designer making a medium design change. The diagram for Reasoning, done by a single designer in a medium design change, is found in Figure 10.5.

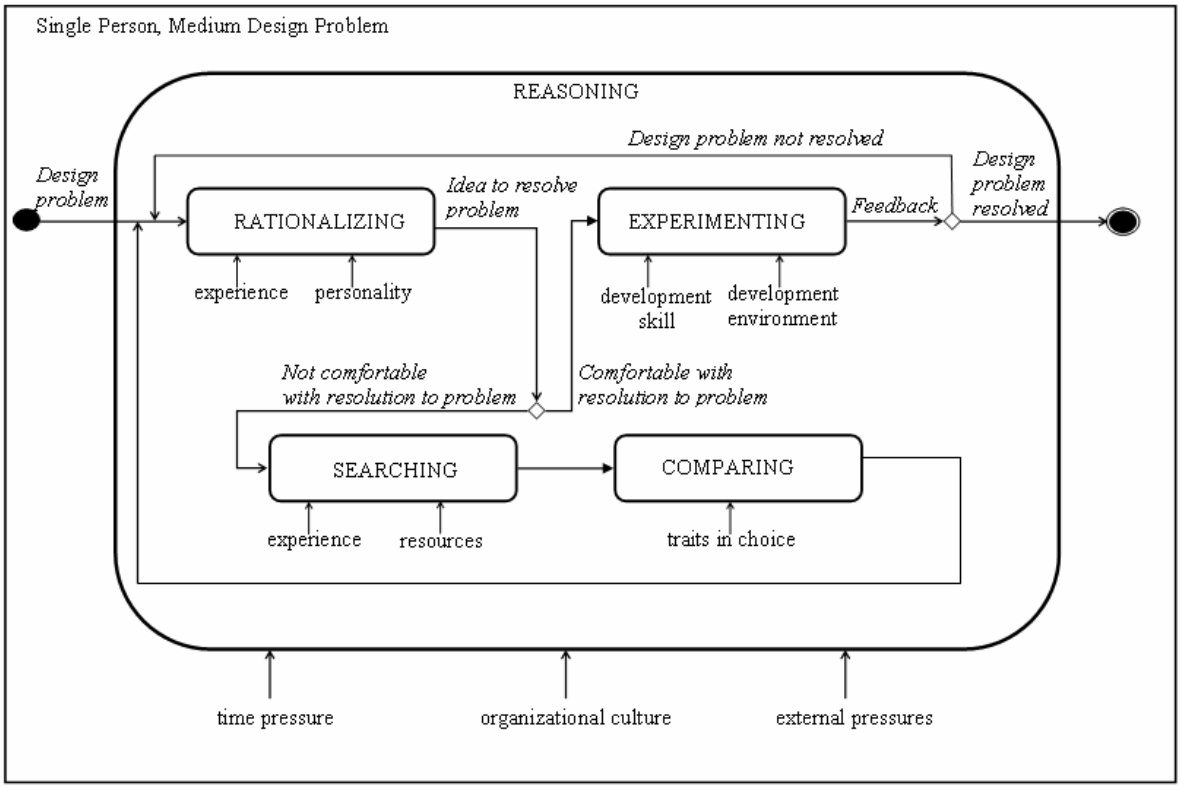

Figure 10.5: Single Designer, Medium Problem Decision Making

The fourth of the five design scenarios is of multiple designers making a medium design change. This employs Reasoning. Rationalizing and Experimenting are used to generate ideas and prototype them, similar to Reasoning in a single designer scenario. The difference here is that when designers try to align their mental models in order to complete a task, they often have much to discuss. The design problem is big enough that their mental models of the existing system are not immediately aligned and they have to discuss the problem domain. This problem domain is often uncertain and this uncertainty leads to different ways to solve the design problem. The group is small enough (often 2-3 people) that they debate solutions in an informal way. Tradeoffs to design solutions are discussed and necessary searches for the "right" answer are conducted with colleagues, on the internet or even in textbooks. This conversation, containing Discussing, Comparing and Searching is called Considering, because it is a natural approach to considering design alternatives. Context variables in Considering are the dynamic of the team and the individual team members' abilities to challenge each other's ideas. Prime examples of this approach are case studies 2.5B and 2.6B where two designers and one customer representative discussed the

customer domain and compared design ideas, while they slowly changed some of the source code. This Reasoning, with multiple designers, is found in Figure 10.6.

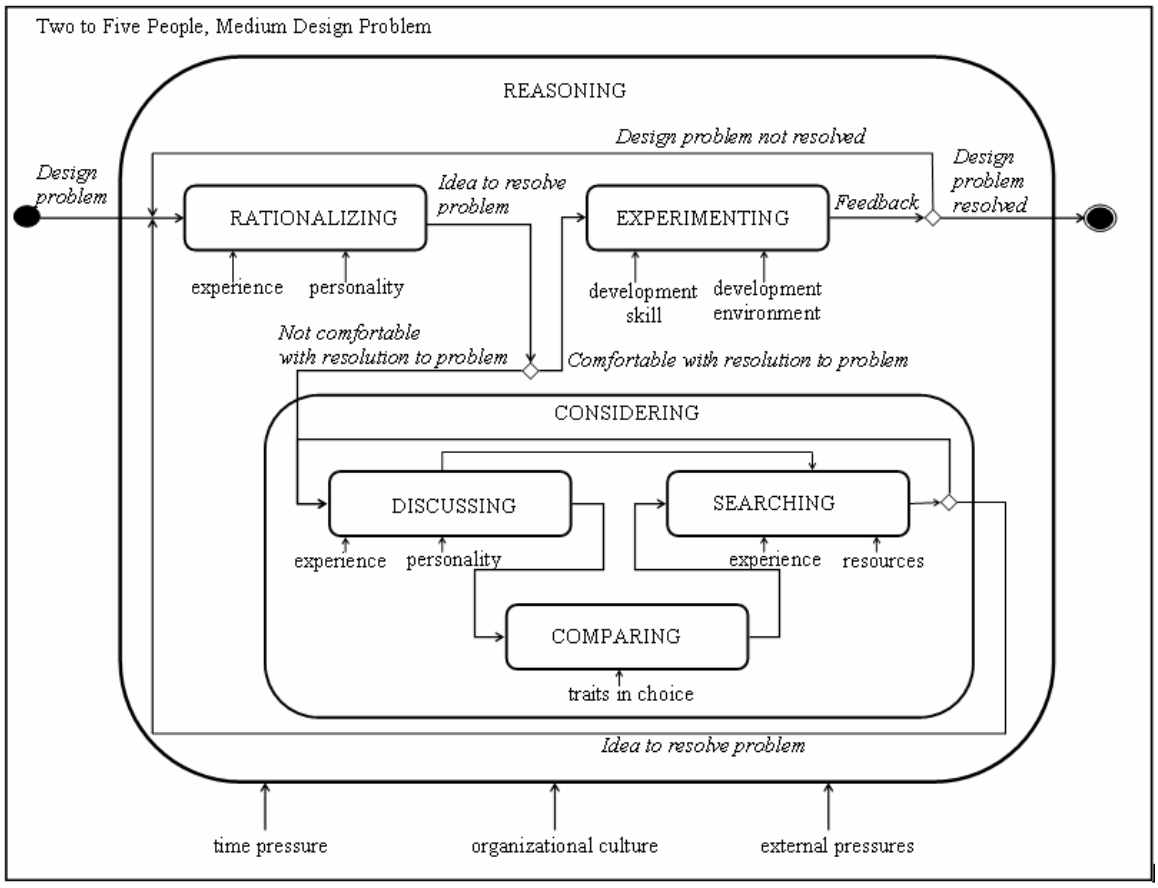

Figure 10.6: Two to Five Designers, Medium Sized Problem Decision Making

The fifth design scenario is one of five to ten people in a large design change. It employs Explaining, primarily because the design change is divided into smaller pieces and the large team is divided into smaller groups. These smaller groups explain their piece of the design change to other small groups in the larger team. In each of the small groups Rationalizing and Considering are used, as in the Reasoning scenario with multiple developers in a medium design change. Discussing is used to communicate about the pieces of the design change between smaller groups in the larger team. There are often dependencies between the smaller design pieces so the discussion is a critical aspect of the Explaining scenario. It is only when these dependencies are resolved that the entire team can move forward. Experimenting does not occur in the Explaining scenario, primarily because the design

changes are so large prototyping is difficult. A prime example is the thick description provided in Section 9.1. The team ended up dividing itself into groups where alternatives were considered and brought results back to the rest of the team. The design decision scenario called Explaining, for a multiple designers working on a large design change is found in Figure 10.7.

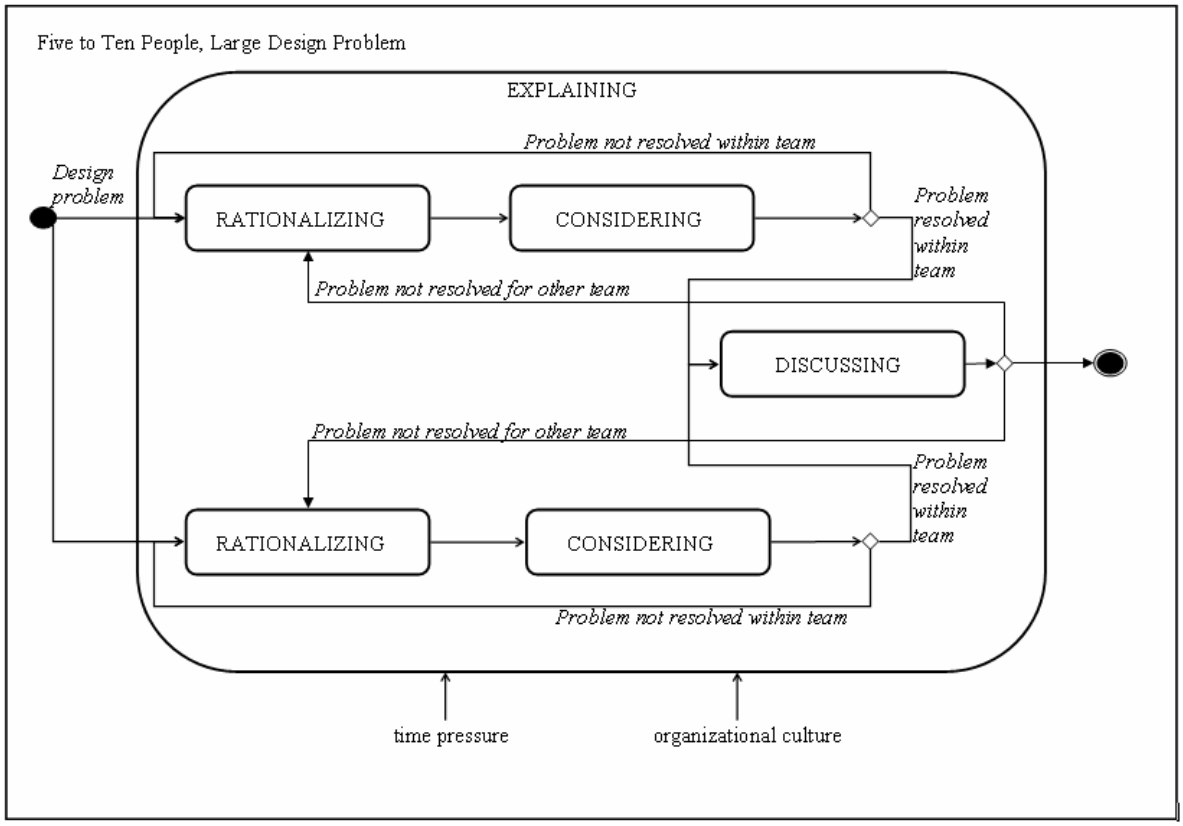

Figure 10.7: Five to Ten Designers, Large Sized Problem Decision Making

I have categorized the 59 decision cases across the three multi-case studies into one of the 5 design decision models. This is shown in Table 10.1. Also shown is the size of the decision problem, showing more use of reasoning (i.e. consequential choice) in medium to large design changes, and more use of appropriating in small to medium design changes. In the first study where Mentors discussed design changes in general I have listed "N/A" for the size of the design change, as Not Applicable, There was only one case study (CS 3.2) that employed Explaining because this was the only case study that involved a large enough group that split into smaller groups to solve a design problem. For more context on each

case study I refer the reader to the case descriptions and quotes provided in chapters 5, 7 and 9 (when discussing a designer's mental model of a system), to Table 5.7, Table 6.2 and Table 8.1 (which summarize the cases in each empirical study), and to the case study summaries found in Appendix A.

Table 10.1 Summary of All Cases and Decision Model Used

Case Study ID	Description of Decision	Decision Model	Size of Design Problem
CS1.1	Decisions reflect understanding of the design.	Reasoning, Two to Five Designers	N/A
CS1.2	Groups should get down to the criteria for decisions.	Reasoning, Two to Five Designers	N/A
CS1.3	Had idea, bought a book, prototyped idea and sold idea to managers.	Appropriating, Single Designer	Medium
CS1.4	Built new test framework based on JUnit and capabilities of .NET and C#.	(some) Reasoning, Single Designer	Medium
CS1.5	Design change is making atomic changes that make design better globally.	Appropriating, Single Designer	N/A
CS1.6	There is no set metric for when teams are "done" with design, have to rely on the comfort level of the team.	Reasoning, Two to Five Designers	N/A
CS1.7	Simple rules and test driven design guide design decisions.	Appropriating and Reasoning, Single Designer	N/A
CS1.8	Customer request requires significant development work, leading to design change to reduce workload.	Appropriating, Two to Five Designers	Medium
CS1.9	Design should be done with short time horizons.	Reasoning, Single Designer	N/A
CS1.10	3 different design decisions, workflow, class hierarchy, technology to transfer data.	Appropriating & Reasoning, Single Designer	Medium (all 3)
CS1.11	New product development	Appropriating, Two to Five Designers	Large
CS1.12	Some design decisions are resolved from previous experience.	Appropriating, Single Designer	N/A
CS1.13	Separate presentation logic from business logic	Appropriating, Single Designer	Small
CS1.14	Make a feature in a legacy	Appropriating, Single	Small

	application easier to use and more well known.	Designer	
CS1.15	Move all caching into one central cache.	Appropriating, Single Designer	Medium
CS1.16	Changing back-end code to match changes in a changed interface.	Appropriating, Single Designer	Small
CS1.17	Change design paradigms after recognizing business mixed with presentation.	(some) Reasoning, Single Designer	Medium
CS1.18	How to do credit card processing, selection of Microsoft products.	Reasoning, Two to Five Designers	Medium (both)
CS1.19	Test improvements made to poor workflow.	Appropriating, Single Designer	Small
CS1.20	Built configuration script into an existing application.	(mostly) Appropriating, Single Designer	Medium
CS1.21	Refactor existing user interface under time pressure.	Appropriating, Single Designer	Medium
CS1.22	Moved logic from a middle tier of the architecture to a stored procedure.	Reasoning, Two to Five Designers	Medium
CS1.23	One-off project developed so that it is useful elsewhere.	Reasoning, Single Designer	Medium
CS1.24	Improve an import structure to handle a large import of data.	Appropriating, Single Designer	Medium
CS1.25	Improve a performance problem. Work on a pet project.	Appropriating & Reasoning, Single Designer	Small Medium
CS2.1A	Bug Fix: Client had a problem over the weekend.	Appropriating, Single Designer	Small
CS2.2A	Bug fix that is going to halt the next release.	Reasoning, Two to Five Designers	Medium
CS2.3A	Feature request to determine data between cashier's terminal and kitchen monitor.	Appropriating, Single Designer	Medium
CS2.4A	Bug fix: developer goes through different versions of the code comparing.	Appropriating, Single Designer	Small
CS2.5A	Feature request in coffee monitor application.	Reasoning, Two to Five Designers	Medium
CS2.6A	Feature request from 3rd	Appropriating, Single	Medium

	party company.	Designer	
CS2.7A	Bug fix.	Appropriating, Single Designer	Small
CS2.8A	Bug fix in a problem with ping	Appropriating, Single Designer	Small
CS2.9A	Feature request, working on a time for the coffee monitor.	Appropriating, Single Designer	Medium
CS2.10A	Feature request so times stop when order is done and to remove items from list.	Reasoning, Single Designer	Medium
CS2.11A	Feature request to implement a replace function	Reasoning, Two to Five Designers	Medium
CS2.12A	Feature request to make configuration form run once and only once.	Appropriating, Single Designer	Medium
CS2.1B	Feature request to incorporate more workflow in design.	Reasoning, Single Designer	Medium
CS2.2B	Bug fix, multiple bills per day for the same customer.	Reasoning, Two to Five Designers	Medium
CS2.3B	Feature request to create filtering on a form.	Reasoning, Single Designer	Medium
CS2.4B	Bug fix in time zones.	Appropriating, (mostly) Single Designer	Small
CS2.5B	Feature request, needs to do more detail in interface.	Reasoning, Two to Five Designers	Medium
CS2.6B	Feature request, how to list objects in a table.	Reasoning, Two to Five Designers	Medium
CS2.7B	Feature request, how to order months using ordinal.	Appropriating, Two-Five Designers	Medium
CS2.8B	Feature request, set rule into charge.	Reasoning, Two to Five Designers	Medium
CS2.9B	Feature request, how to load beer on a truck.	Reasoning, Two to Five Designers	Large
CS2.10B	Bug fix, capitalization looks careless	Reasoning, Two to Five Designers	Medium
CS2.11B	Feature request, adhoc rules fire charges but are not based on a built in rule.	Reasoning, Two to Five Designers	Medium
CS2.1C	Feature request requiring interaction with a system with no SDK.	Reasoning, Single Designer	Medium
CS2.2C	Feature request to make a message pop up to a user.	Appropriating, Single Designer	Small
CS2.3C	Bug fix	Appropriating, Single	Small

		Designer	
CS2.4C	Whether or not to use .NET	Reasoning, Two to Five Designers	Large
CS2.5C	Bug fix to make inputs more constrained by creating a drop down box in.	Appropriating, Single Designer	Small
CS3.1	Personalization in the first launch of the new portal.	Reasoning, Two to Five Designers	Large
CS3.2	Portal Architecture	Explaining, Five to Ten Designers	Large
CS3.3	Which portal search tool to use	Appropriating, Single Designer	Medium
CS3.4	How much information had to be searchable for the first launch of the portal.	Appropriating, Single Designer	Medium
CS3.5	The choice of a web content management tool.	(some) Reasoning, Two to Five Designers	Medium
CS3.6	How navigation would look on the new portal.	Appropriating, Two to Five Designers	Medium

10.5 Chapter Summary

The three empirical studies contributed arguments towards two main conclusions of this research. The arguments were consistent with each other. The alignment of the arguments across the three studies give weight to the conclusions made, as per triangulation [Patton02]. Most of the lessons learned in each study were specific to that study but the idea that time pressure imposed the use of serial evaluation did occur in two of the three studies. This chapter provided five models of design decision making showing three approaches: Appropriating, Reasoning and Explaining.

CHAPTER ELEVEN: VALIDITY & RELIABILITY

"Validity is a property of knowledge, not methods. No matter whether knowledge comes from an ethnography or an experiment, we may still ask the same kind of questions about the ways in which the knowledge is valid." [Patton02, p587]. To address validity in this research I used multiple approaches, depending upon the empirical study I was validating. I define these approaches as needed and address each study individually. To begin, the validity of a finding is defined as, "the degree to which the finding is interpreted in a correct way" [Patton02, p94]. Reliability is, "the degree to which the finding is independent of accidental circumstances of the research" [Patton02, p94].

11.1 Evaluating Design Decision Making via Interviews

The first empirical study involved approaches to validity that were typical of a controlled experiment or a positivist view of case studies [Patton02, Yin02]. I address construct validity, internal validity, external validity and threats to validity of the first empirical study.

11.1.1 Construct Validity

The critical decision method was used to capture components of a design change. The critical decision method used in examining decision making is a well used tool [Klein89]. The assumption that a design change is a critical incident may be argued. However, the purpose of the critical decision method is to examine cognitive skills used in decision making about a critical incident, not just the critical incident itself. Effecting design change is a cognitive skill requiring decision making.

Concerning the re-coding and the frequencies, I have stated that I did not go back to re-code when a new code was added and recognized that this could potentially impact the frequencies that I present. This is not considered a large concern, however, because the codes that were added were ultimately not dominant in frequencies or conceptually, over the entire coding process. For example, the code **Time** was added to handle words and phrases such as "now" or "at this time". Even after adding this code, it did not emerge as

dominant in the entire coding process, especially in comparison to some of the more dominant codes discussed in this thesis (e.g. **Right vs. Wrong, Better vs. Worse**, **Model**, **Knowledge**). It is for this reason that the frequencies presented here are still worthy of examination.

11.1.2 Internal Validity

The results of the first empirical study are validated by addressing the interpretations of the answers to the interview questions. I compared the codes from the Specific Relational Coding to summaries of each question which I had coded. That is, I summarized the responses of each interview question for each interview participant in my own words, then I coded that summary.

An example of this validation is now provided. In one interview I asked a developer where he attained the knowledge to implement the design change he was discussing, as found in the probing question on knowledge, in Table 4.1. The developer responded as follows:

> *"I had used XML and I think I played around with XSLT a little bit… I went out and bought a book, the first book on XSLT that came out, a very nice book, and I just started devouring that book and started trying it out, seeing what worked… I bought the book to implement the design idea." Anonymous Agile developer*

The SRC (Specific Relational Coding) for this quote produced the relational codes provided below. Definitions of these codes are provided in the Appendix.

[Experiment – Technology] & [Search – Knowledge]

I then produced a summary of the entire case study in which this quote exists. For the response to the knowledge question the case study summary is as follows:

> *"He knew a little bit about a potential solution, went and looked it up, bought the book, read it and tried it out." Field Notes*

From this summary I produced the following relational coding:

[Search – Knowledge] & [Experiment – Knowledge]

Figure 11.1 shows an example of this matching process. I wrote a case study summary of an interview participant, and then coded the summary. Note the codes at the bottom of Figure 11.1 that read "experiment – alternatives, comms – alternatives (i.e. **Communications – Alternatives**) and BvsW – alternatives (i.e. **Better vs. Worse – Alternatives**. I searched the original Large Window, Relational Coding for these codes. As shown in Figure 11.2, the code alternatives – B vs. W (i.e. **Alternatives – Better vs. Worse**) was in the original relational coding. Thus there was a match in the topics I found. The case study summary and coding of that summary occurred *after* I had first coded all of the interview participants, and 4-6 weeks apart, so memorizing a code was not considered a large issue.

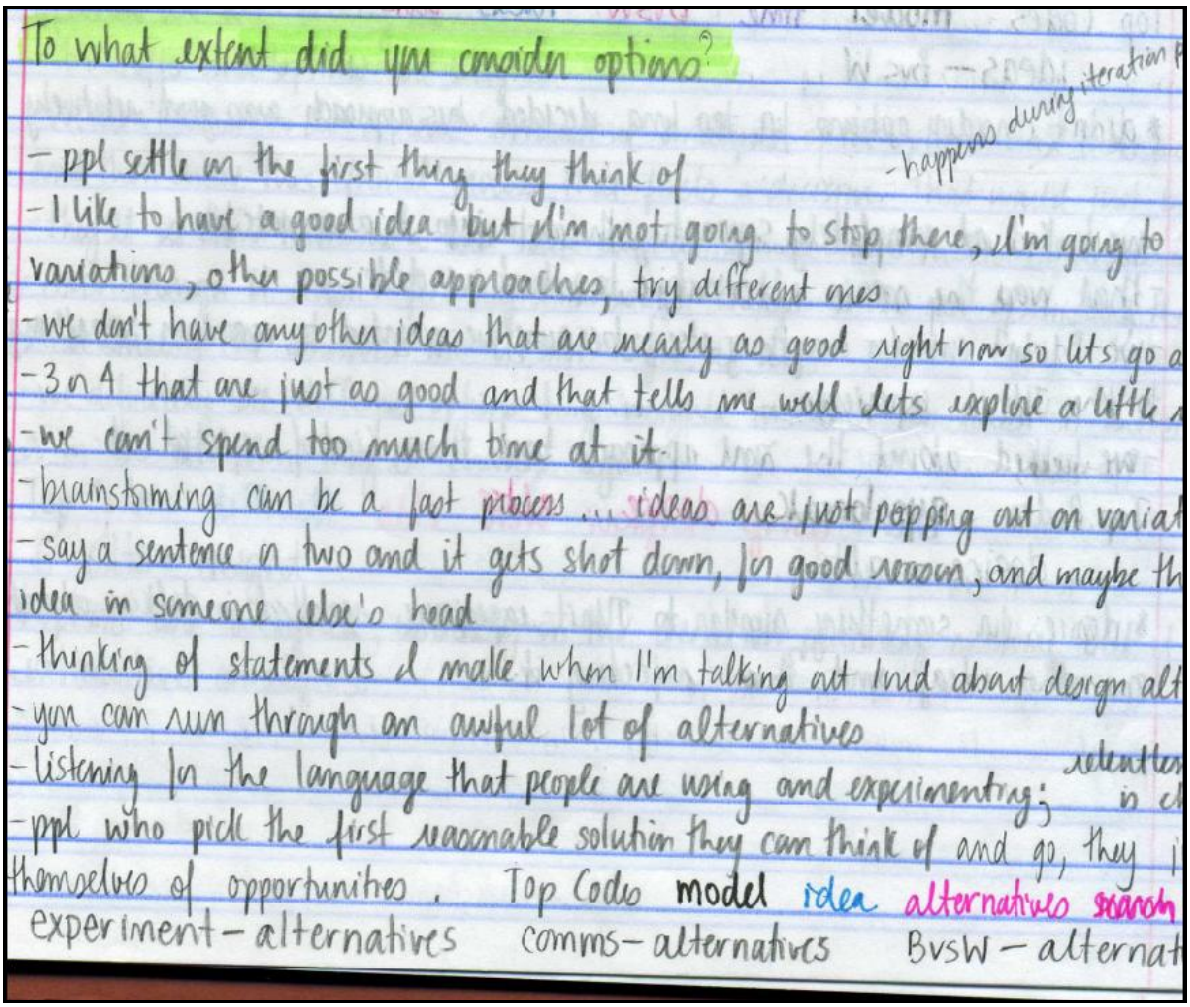

Figure 11.1 Scanned Photo of Case Study Summary for a Question

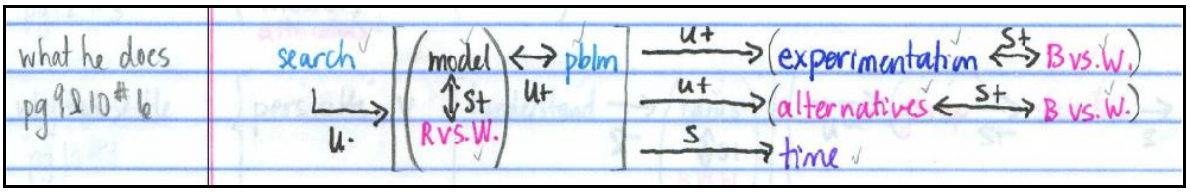

Figure 11.2 Scanned Photo of Relational Coding used to Compare to Coding of Case Study Summary

I search the Specific Relational Coding of each interview participant for matching codes to validate the summary of the question. A match meant that I found the codes from the coded summary in the codes generated by Specific Relational coding, for each of the seven questions asked. All the case study comparisons showed at least 5 out of 7 matches between the coded summaries and the Specific Relational Coding of each interview. Typically when there was a mismatch it was because the Specific Relational Coding found codes that did exist while the coded question summary showed what was not in the response to the interview question. For example, if time pressure was not an issue in a decision the Specific Relational Coding contained other codes representative of what was an issue in a decision, whereas the coded summary contained the code **!Time**, to show that time was not an issue. Table 11.1 shows the mismatches that occurred for each question for each interview participant and the total number of matches for each interview participant.

11.1.3 External Validity

Given that I conducted case studies, not experiments, the external validity relies on generalization to theory (analytical generalization) and not statistical generalization. Almost all (23/25) of the case studies presented align with the theory that the strength of a mental model impacts the approach to decision making. All of the case studies presented align with the theory that designers do not consistently strive for an optimal design, or a boundedly optimal design.

11.1.4 Threats to Validity

There are three threats to validity that merit discussion. The first is that most of the interview participants were familiar with, used or discussed agile methods. It can be argued that the principles defined in the agile manifesto [Agile07] align with the definition of naturalistic decision making [Klein98]. As a result, the first study may include a larger

emphasis on naturalistic decision making versus rational decision making. To mitigate this the observations and participation studies were conducted.

Table 11.1: Mismatches and Total Number of Matches between SRC and Coded Summary

Interview	Decision	Cues	Knowledge	Altern-atives	Experi-ence	Time Pressure	External Goals	# Matches
1							N/A	6/6
2								7/7
3					X	X		5/7
4								7/7
5							X	6/7
6						N/A	N/A	5/5
7						X	X	5/7
8					X			6/7
9								7/7
10		X	X					5/7
11								7/7
12						X		6/7
13				X			X	5/7
14						N/A	X	5/6
15								7/7
16							N/A	6/6
17						X	X	5/7
18							X	6/7
19								7/7
20	X						X	5/7
21				X	X			5/7
22								7/7
23			X			X		5/7
24								7/7
25								7/7

The second issue is that I do not report the types of systems that were discussed in the design decision (except to explain the context of the system). The primary reason for this is that the type of system was not the topic of analysis, individual design changes were. Because there is not extreme variation in the types of systems discussed, (e.g. personal projects and critical systems) because most of the systems were small to medium sized projects (i.e. many were agile projects) and because the interviews addressed design change not an end to end system, the issue of the type of system is seen as a small threat to validity.

The last threat to validity is that the results are subject to the weaknesses of retrospective interviews. Interview participants reported their results as they remembered the design change. Thus, the results are subject to their recollection. In order to validate these results they are compared against the results of the observations and participation. Throughout this thesis I have shown consistencies among the interview results, the on-looker observation results and the participatory observation results.

11.2 Evaluating Design Decision Making via On-looker Observations

The second empirical study involved approaches to validity that were typical of a positivist approach to case studies [Yin02, Patton02, p94]. I address construct validity, internal validity, external validity and repeatability of the second study.

11.2.1 Construct Validity

First, I justify the use of observations as the approach to data collection and my presence during the observations. I selected observations as the approach to data collection because it was the second step in the large empirical study that is this thesis. Thus I am not too concerned about *why* I chose observations for this study. Perhaps more important is my impact on the observed environment. During the observations I recorded 19 comments that indicated some impact I had on the environment, either positive or negative. In almost every case my presence was noticeable. I recorded two comments that indicated developers "forgot" that I was present during their work. In addition, I recognize that the observations were only a glimpse into the life of an organization. While the observations occurred for 9-10 days, factors that affect the conclusions may have occurred before or after my observations, without my knowledge. Future long term studies would be suitable to validate the results.

11.2.2 Internal Validity

I addressed internal validity by minimizing the codes in content analysis and using explanation building [Yin02]. Previous experiences with content analysis with a larger number of codes led to inconsistencies in inter-rater coding because of the cognitive load placed on the second coder. I reduced the number of codes significantly and compared

coding and results across cases frequently to ensure the propositions were consistently supported through the codes [Yin02].

11.2.3 External Validity

Given that I conducted case studies, not experiments, the results are generalized to theory, not statistics. The results are based on experiences from the software industry, not from student experiences. The results speak to design decision making and real options analysis in small non-safety critical organizations that have tendencies towards principles of the agile manifesto [Agile07].

11.2.4 Reliability

I addressed reliability and repeatability using a consensus estimate for our measure inter-rater reliability [Stemler06]. I compared results of analysis performed by three authors for 8 of the 28 case studies (29% of our data) We achieved a consensus estimate of 77%, 76% and 75% between each pair within the three authors, comfortably above the recommended rate of 70% [Stemler06].

11.3 Evaluating Design Decision Making via Participatory Observations

For the third empirical study I address validity by discussing three points: authority [Schwandt01] and representation [Schwandt01] and the strength of my technical understanding of the project at E-Solutions Inc. I do not consider my interpretations of the interpersonal interactions to be the *only* interpretation. From an auto-ethnographic viewpoint, it was my intent in Section 9.1 to, “unite ethnographic (looking out-ward at a world beyond one’s own) and autobiographical (gazing inward for a story of one’s self) intentions.” [Schwandt01, p.13] Thus, while I recognize there are alternative perspectives on the events that occurred at E-Solutions Inc. during May 2006, and while I recognize my interpretations of these events do not produce a single objective representation of the events, I also emphasize that such a representation was not the purpose of the third empirical study. I was the only person on the team in the role of researcher-as-decision-facilitator and from an auto-ethnographic perspective, my illustration of the decision

making and attempts to change decision making have much merit, especially in light of having conducted the first two empirical studies.

To address the points of authority and representation, and to confirm my technical understanding of the project at E-Solutions Inc., I asked 5 members of the team who worked with me on the portal project to read Section 9.1 and answer three questions addressing their thoughts on the accuracy of the report, their recollection of the events that occurred and their comfort with the events being reported. They answered these questions three times during their reading of Section 9.1. The first time was after reading the introduction of Section 9.1, found from pages 148-155 of this thesis. The second time was after reading the technical description of the project, found from pages 155 - 158 of this thesis. The third time was from pages 158-166 of this thesis to the end of the report. I asked them to fill out the questionnaire three times to give them multiple opportunities to comment on my report. Each time, the questions were provided in the format shown in Figure 11.1. The readers also provided some comments to me via email as they returned the report.

The reader's responses, provided in Table 11.2, show agreement with the report, an adequate recollection of the events that occurred and a comfort with the results being reported. This last question was provided to be ethically thorough.

Comments from the readers after reading the report are below.

Comment 1: "Your report was very accurate on the portal jam sessions so much so that it was scary. I am glad you do not witness any of my other work. Your report transported me back in time to May 2006. I could very well recollect how badly those meetings ran."

Comment 2: "It is hard [to determine accuracy of the first section] not knowing the role I played [in the report], but it seems like an accurate account."

Comment 3: "When I read the account of the deployment, I realize what a dysfunctional disaster it was. This report highlights why even the smallest changes required enormous conversation."

Questions for the Reader

Please answer the questions below by circling word that best describes your impressions after reading the first part of this report. Feel free to add your candid comments to further qualify the word you circled.

1. Thus far, this report is an accurate account of the changes we tried to make to the portal jam sessions.

Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree

Comments:

2. I remember the events that occurred in May 2006:

Not Well at All	Not Well	Neutral	Well	Very Well

Comments:

3. Given my involvement in the reported events and the anonymity conveyed in the report, my personal comfort level with this report is:

Not Comfortable At All	Not Comfortable	Neutral	Comfortable	Very Comfortable

Comments:

Figure 11.3: Questionnaire Provided to 5 Members of the Portal Team

Table 11.2: Responses from Readers after Reading Section 9.1

Question	Section of Report	Reader 1	Reader 2	Reader 3	Reader 4	Reader 5
1 Accuracy	1	Strongly Agree	Neutral	Agree	Strongly Agree	Agree
	2	Strongly Agree	Agree	Agree	Agree	Agree
	3	Agree	Agree	Agree	Agree	Agree
2 Memory	1	Well	Well	Neutral	Very Well	Well
	2	Well	Well	Neutral	Very Well	Very Well
	3	Well	Well	Neutral	Very Well	
3 Comfort	1	Very Comfort-able	Neutral / Comfort-able	Very Comfort-able	Very Comfort-able	Very Comfort-able
	2	Very Comfort-able	Very Comfort-able	Very Comfort-able	Very Comfort-able	Very Comfort-able
	3	Very Comfort-able	Comfort-able	Very Comfort-able	Very Comfort-able	Comfort-able

Comment 4: "My opinion for what its worth is that this project was a mess. The caboose (IT) was trying to lead the train. It would have been better if there were functional guidelines to drive the decision making, what I call the, 'I need to do x'. For example: I need to edit the portal content online and in place. This drives IT to propose solutions with cost and time. For example, I can not tell you that you must buy $3 coffee every day for the next year. No business analyst and a lack of project management just about killed this project."

Comment 5: "The account does not portray or account for some of the background to arrive where we did with the meetings. i.e. Context is missing.

Comment 6: "It is interesting from my perspective to hear about interactions that I had no knowledge about. It provides a more holistic view of events."

Comment 7: "Section one was a tad harsh but sometimes the truth hurts. I am okay with everything that is in the report. I would have written it differently in some cases. People do see events differently and it was good to get a different viewpoint. As the story starts in the

middle of a year long process it does lack some context but not sure how much more would add value. You have tried to capture the mood and process is a short few pages and that is hard. It is hard to guess why people said or did without a lot more detail."

The project discussed in Section 9.1 was challenging on many levels for many people. Regardless, the comments above and the responses to the questions support the account provided in Section 9.1 and can be used to validate the results generated from Section 9.1.

11.4 Comparing Results to the Literature

As a last approach to addressing validity I examine my results as they compare with concepts from the literature. There were numerous concepts and decision making approaches mentioned in Chapter 2, and I highlight these in bold as I discuss if the results of this thesis are aligned with these concepts and approaches.

For the general concepts from Chapter 2, there was the concept of problem solving and recognizing software design as problem solving. This research is aligned with the ideas of **problem structuring** and **continual restructuring** by showing that alternatives to decisions were considered when problems were more ill-structured, or the designer had a more open mental model of the design problem [Zannier07b]. There is some debate as to the types of knowledge required to produce "good" design [Adelson85, Sharp91]. This work does not resolve this debate. Instead, this thesis recognizes that various types of design knowledge are used to structure a problem, for example, external factors such as market pressure, knowledge of design heuristics, and learning about the software domain (e.g. Zannier07b, and structural and market knowledge addressed in Chapter 7). Because this thesis supports the idea of continual restructuring and problem structuring, the ability to cleanly separate the design problem from the design solution is not supported, and, in my opinion this impacts the ability to create design knowledge categories entirely cleanly.

This work does not align with the idea of e**xplicitly capturing of design rationalizations** through the difficulties faced at E-Solutions Inc. and the use of the QuickPlace tool. **Mental modeling** and **mental simulation** are explicitly referenced throughout this thesis

by showing varying levels of strong or open mental models. In a similar vein the idea of **shared cognition** was apparent through developers trying to align their ideas with each other when working together. The concept of **designer expertise** was implicitly recognized by discussing the differences between some designers who were observed. Some designers had a good understanding of design patterns or had a recognizable desire to pay attention to detail in their work, whereas other developers did not. **Preferred evaluation criteria** was not explicitly referenced but is understandable in light of the results showing that developers appropriate a design solution. **Group think** was not abundantly apparent, even in the group work that was conducted at E-Solutions Inc.

For the decision theories described in Chapter 2, this research is consistent with some general aspects of **Multi-Attribute Decision Theory** such as consequential choice. However, specific adherence to the method of Multi-Attribute Decision Theory and to **Real Options Theory** is not aligned with this research. Explanations for this may be the type of design problems discussed, but also the subjective and informal manner in which software designers discuss design. This work is consistent with **Requisite Decision Theory**, which operates with a *sufficient model* that is *shared* by members of a team working on a problem together. This research does not show significant evidence of **Prospect Theory** which states that a decision maker will select more certain situations when possible gain is large, even if the certain situation results in a smaller gain than the alternative [Kahnema79]. This research also does not show significant evidence of **Signal Detection Theory** [Cooksey95]. An example of using Prospect Theory in software design is copying and pasting code from one section of the system to another, because the developer knows the code works and does not have time to consider refactoring the common code. **Elimination by Aspects** says that each alternative to a decision has a set of aspects, repeatedly chosen with a probability that is proportional to its weight. [Tversky72]. This is understandable based on this research, although was conducted implicitly by the designers. There was no explicit reference to **Social Judgment Theory** in this research. **Heuristics and Biases** were definitely apparent throughout this research. It can be argued that designers used representativeness, availability and anchoring and adjustment when making their choices, and this is incorporated in the “traits of choice” context variable in the Comparing activity found in

the models in Chapter 10. Similarly, bolstering and deemphasizing from **Dominance Structuring** was can be seen as a trait of choice, and **Explanation Based Decision Making** was identified as a potential model of describing design decision making. **Participatory Decision Making** is an understandable approach to decision making, in light of the experiences at E-Solutions Inc, but after these experiences it is also more challenging to implement than would have originally been anticipated. Finally, components of **Recognition Primed Decision Making** are applicable to this research through the concepts of serial evaluation, situation assessment and satisficing [Klein98].

11.5 Chapter Summary

This chapter addressed the validity of each case study from various perspectives. The first empirical study took a more positivist look at validity. The second empirical study took a slightly positivist, slightly interpretivist look at validity. The third empirical study took an interpretivist look at validity.

CHAPTER TWELVE: CONCLUSIONS & FUTURE DIRECTIONS

This qualitative empirical investigation provides a set of results and a set of design decision models that together describe software design decision making. To review, the conclusions are as follows: **software designers appropriate design solutions about as often as they consider alternatives to design solutions**. **Software designers consider alternatives to design solutions when they have an open mental model of a design problem or when they are working in small groups discussing the design problem**. **When designers are working alone they are more apt to appropriate a single design solution to a design problem**. That is, they apply a design solution that "just fits" or is "suitable". **Time pressure and external goals encourage the use of appropriating a design solution to a design problem**. Finally, **rational decision making and naturalistic decision making are somewhat aligned with design decision making, real options theory is marginally aligned with design decision making and explanation based decision making is partially aligned with design decision making**. These results emerged by comparing the concept optimzing to the concept satisficing (as rooted in rational and naturalistic decision making respectively), by merging the idea of problem structuring with the idea of mental modelling, by using successive case studies to confirm or refute the emerging ideas, and finally by shaping shaping and re-shaping the word choice presented in the conclusions.

Purely descriptive research is often challenged, in traditional science communities, as lacking a "so what?" component. Thus, I have identified 3 topics in the software engineering community which are affected by the results presented in this descriptive research.

First, there is on-going debate concerning whether or not software development is an art or a science. To a large extent, this debate is merely anecdotal [McConnell98], but the debate occurs implicitly in forms other than anecdotal. For example, agile methods highlight the importance of managing ambiguous and changing requirements, prioritize individuals and interactions over process and tools [Agile07], and thus arguably follow the software-as-art philosophy more so than the software-as-science philosophy. The momentum that followed

the manifesto [Agile07], the published and un-published debates surrounding agile versus non-agile in the software engineering community (e.g. [Melnik06]), and the resulting pursuits to empirically validate agile approaches as more than ad-hoc procedure, are arguably versions of the software-as-art versus software-as-science debate. As another example, the identification of software development as a wicked problem [Poppend03, Rittel73], or the empirical support identifying problem structuring as a component of software development [Guindon90a, Guindon90b, Zannier07b] shows work is still being performed to argue for software-as-art, rather than software-as-science. Finally, the idea that qualitative research on the social side of software is relatively "new" in software empiricism shows work is still being performed to argue for the "softer" aspects of software development.

This empirical investigation concludes that software designers appropriate design solutions approximately as often as they strive for an optimal or even boundedly optimal design solution. This is in part because software design is predominantly represented in subjective terms. Optimal solutions and boundedly optimal solutions are often not even known, so how can they be achieved? It can be argued that this topic is a "done deal", that the ideas of design-as-art, wicked problems, and the like, have permeated a significant portion of the software engineering community and the point is moot. Were this the case, I believe the varying forms of the debate between design-as-art and design-as-science would not exist or would not carry the momentum they do carry. Alternatively, work in the area of design-as-art would carry momentum equal in proportion and stature to work that implicitly supports the design-as-science argument [McConnell98]. An issue is that the software community has not determined which is "correct" – design as art or design as science – or which is "desired", potentially complicating communication and the definition of plausible goals in a research program. Thus the manner in which a piece of research contributes to one side or the other should be made explicit when possible. The impact of this research on the software engineering community is therefore to provide strong empirical evidence for the design-as-art argument, and to provide support for other empirical studies identifying the use of problem structuring [Simon73], and continual restructuring [Adelson85, Guindon90b] in software design.

Second, the existing work in decision making, to be applied to software development, has been in a sense, top-down. There are recommendations for how decision making should occur [Sullivan99, Boehm00], and there are a plethora of decision making approaches that can be used as platforms for software decision making (e.g RDM in decision support tools). However, there are very few studies [Curtis88, Guindon90a] that examine or address decision making as it naturally occurs, or in a sense, bottom-up. A significant contribution of this work to the software engineering community is to begin to resolve this void. The results of this thesis show the limitations of applying rational decision making, naturalistic decision making and real options theory to certain decision problems. The results also show potential for the use of explanation based decision making, in describing software design decision making. In addition, the results show areas that are still uncertain – knowledge acquisition and time pressure – areas that are important in completely understanding design decision making. The impact of this research on the software engineering community is therefore to provide a bottom-up approach to understanding design decision making, which can be used in conjunction with top-down approaches in at least three specific directions. The first direction is to address when and how decision making approaches such as real options theory naturally align with descriptions of software engineering. The majority of the design decisions discussed in this research did not involve budget constraints, or long-term project timelines, and this is perhaps a natural area for real options analysis to align with design decision making. At this time, an open problem is how and when to apply real options theory in software design because there is a lack of experience reports on the topic. The second direction is to resolve the inconsistencies in knowledge acquisition and time management in the course of design decision making. The third direction is to apply change management guidelines to effect change in design decision making.

Third, two important areas of software engineering research are affected by this empirical examination. The areas are decision support tools and software metrics. The second result of this research, that the approach to decision making is affected by the strength or openness of a designer's mental model, highlights the cognitive aspect of design decision making. The result that environments that foster casual communication naturally employ

consequential choice more than serial evaluation, highlights the social aspect of design decision making. For decision support tools, these results suggest that decision support tools should be interwoven with tools that manage existing models of applications under design, should accommodate serial evaluation when those models are well-defined, and should support the socio-cultural context in which consequential choice and decision making naturally occurs. For example, tool support for decision capture could focus on single display groupware support (multiple users interacting simultaneously on a single device) instead of developing personal tools for capturing design rationales. For software metrics, these results suggest that metrics should measure an aspect of design *relative to* a designer's past experience, knowledge of the system, type of design problem on which s/he is working, and perhaps even personality. Perhaps this occurs in the form of peer evaluation, or another context-aware evaluation. An example of the points of evaluation could be adherence to a specific and articulated set of design heuristics recommended by the company, meeting a deadline, improving the accuracy of time estimates for task completion and correct implementation of an algorithm. In addition, the social context in which consequential choice naturally occurs suggests that there is more merit to taking metrics in (and possibly during) group design discussions than in design conducted by single designers on small problems. A potential metric could be the number of alternatives considered, and or, whether the alternatives were supported by personal experience or design heuristics or both. The impact of this research is thus to provide points of evaluation for existing decision support tools and software metrics.

There are four paths I can identify on the topic of design decision making, that merit further research. The first path is one of validation. The second path is one of tool support. The third addresses process support. The fourth examines the quality of a decision. For validation, the five models presented in Chapter 10 can be used in future case studies of design decision making. I recommend think aloud protocols for software developers working alone, and audio recording team meetings, especially informal ones. Results from such case studies can be compared to the decision models presented in this thesis to support, refute or refine the models.

At this time an open issue in software development is the gap between recommendations and tool support for decision making, and descriptions of how designers work and make decisions. To address the topic of tool support for decision making, I recommend four steps. First, I recommend the decision models be evaluated. Second, I recommend a survey of decision support tools be performed to determine the assumptions and foundations of such tools. I would then compare the assumptions and foundations of decision support tools to existing descriptions of decision making, to determine what ideas from each area intersect. Before introducing a decision support tool into a software organization I recommend a plan for change management to handle the ideas built into the tool that do not intersect with descriptions of design decision making.

At this time an open issue in software development is research in software process change management. To address process support, I recommend surveys of team dynamics and group work to determine where ideas in this field intersect with descriptions of software design decision making. I believe decision making can be supported by process, not tool support, and given this I would outline a plan for effective group interactions that encourages considering alternatives. In addition I would plan for any changes introduced by this process.

Finally, if the topic is to evaluate the quality of a design decision, I believe the challenge is significant. First, a culture must be established where considering alternatives to decisions is valued *as much as* completing a decision. This is likely a significant cultural step. After this, qualitative context-aware metrics should be developed that can be assigned to some cost value (e.g. via developer time, recognizing developer expertise). This is a significant step as well. From this, the quality of a decision can be defined via minimizing cost.

To summarize, the contributions of this research are:

- To provide empirical evidence of Appropriating in software design decision making.
- To provide empirical evidence showing the way a software designer understands a design problem impacts the way s/he makes decisions within that design problem.

- To provide empirical evidence showing the way a software designer approaches a design decision can be affected by the work environment in which the decision is made.
- To provide strong empirical evidence for the design-as-art argument.
- To provide support for other empirical studies identifying the use of problem structuring [Simon73], and continual restructuring [Adelson85, Guindon90b] in software design.

The implications of these contributions are:

- Decision support processes and tools should recognize the cultural context and the design context in which a design decision occurs.
- Changes to decision making should recognize the social component of decision making and the details of implementing change in a software team.

I conducted an empirical investigation of software design decision making. This investigation consisted of three multi-case studies. The first result was that software designers appropriate design solutions approximately as often as they strive for an optimal design solution or even a boundedly optimal design solution. The second results that the approach taken to decision making was affected by the strength and openness of the designer's mental model. This research also provided a set of smaller results, one being that time pressure imposed the use of serial evaluation in decision making, and another being that consequential choice occurred more often in environments that fostered continual communication. This research supports the argument for software design as art, it begins to fill the void in bottom-up research in design decision making, and it makes specific recommendations for decision support tools and software metrics that recognize the inherent tendencies of software design decision making. Finally, this research is a tangible contribution to qualitative inquiry in software engineering research.

BIBLIOGRAPHY

[Adelson85] Adelson B, Soloway E; "The Role of Domain Experience in Software Design"; IEEE Trans. Software Engineering, V SE-11 No.11 November; 1985

[Agile04] XP Agile Universe Conference Web Site Http://www.xpuniverse.com Last Visited May 2007

[Agile07] The Agile Manifesto http://agilemanifesto.org/ Last visited March 2007

[Ahmed03] Ahmed S, Wallace K, Blessing L; "Understanding the Differences Between How Novice & Experienced Designers Approach Design Tasks"; Research in Engineering Design 14, 1-11; 2003.

[Alexander79] Alexander, C; The Timeless Way of Building, Oxford University Press New York 1979

[Arishol04] Arisholm E, Briand L, Foyen A; "Dynamic Coupling Measurement for Object-Oriented Software"; Proc. 26th Int. Conf. on Software Engineering (ICSE); 2004

[Arishol04b] Arisholm E, Sjoberg D.I.K; "Evaluating the Effect of a Delegated versus Centralized Control Style on the Maintainability of Object-Oriented Software" Proc. 26th Int. Conf. on Software Engineering (ICSE); 2004

[Basili86] Basili,V.R, et al "Experimentation in Software Engineering"; *IEEE Trans. Software Engineering* SE-12 7 July 1986.

[Basili92] Basili; "The Experimental Paradigm in S/W Eng." Proc. Int. Workshop. Experiment. S/W Eng. Issues; V706 1992.

[Basili99] Basili V.R, et al; "Building Knowledge through Families of Experiments"; IEEE Trans. Soft. Eng. V 25 No 4, 1999

[Basili96] Basili V.R, Briand L.C, Walceilio L; "A Validation of Object-Oriented Design Metrics as Quality Indicators"; IEEE Transactions on Software Engineering; V 22 No 10, pp 751-761, October 1996

[Beach93] Beach L.R; "Image Theory: Personal & Organizational Decisions"; In: Klein GA, Orasanu J, Calderwood R, Zsambok CE, editors. Decision making in action: models and methods. Norwood, NJ: Ablex Publishing Corporation; 1993.

[Beck01] Beck K, Fowler M; Planning Extreme Programming, Addison

Wesley, 2001

[Beckhard69] Beckhard, R. 1969. *Organization Development: Strategies and Models*, Addison-Wesley, Reading, MA.

[Beecham03] Beecham S, Hall T, Rainer A; "Software Process Improvement Problems in Twelve Software Companies: An Empirical Analysis"; Empirical Software Engineering V 8 No 1 pp 7-42, March 2003

[Bellotti96] Bellotti V, Bly S; "Walking Away from the Desktop Computer: Distributed Collaboration and Mobility in a Product Design Team"; Proceedings of the 1996 ACM Conference on Computer Supported Cooperative Work, Boston, Mass. USA; pp209-218 1996

[Boehm00] Boehm B.W, Sullivan, K.J; "Software Economics, A Roadmap"; International Conference on Software Engineering, Proceedings of the Conference on The Future of Software Engineering, Limerick, Ireland Pages: 319 – 343, 2000

[Bratthall02] Bratthall L, Jurgensen M; "Can You Trust a Single Data Source Exploratory Software Engineering Case Study?" Empirical Software Engineering; V7 No 1, p9-26, March 2002

[Brehmer88] Brehmer B, Joyce G.R.B; "Human Judgment: the SJT View"; Elsevier Science Ltd; 1988

[Briand93] Briand L.C; Morasca S, Basili V; "Measuring and Assessing Maintainability at the End of High Level Design"; Proceedings of the Conference on Software Maintenance; IEEE Computer Society, Washington, DC, USA, pp88-97, 1993

[Briand96] Briand L.C., Morasca S, Basili V.R; "Property-Based Software Engineering Measurement"; IEEE Transactions on Software Engineering; V 22 No 1, January 1996

[Briand98] Briand L.C, Daly J, Porter V, WustFranhofer J; "A Comprehensive Empirical Validation of Design Measures for Object Oriented Systems"; Proceedings of the 5th International Symposium on Software Metrics, p246, 1998

[Briand99] Briand L, Arisholm E, Counsell S, Houdek F, Thevenod-fosse P; "Empirical Studies of Object-Oriented Artifacts, Methods and Processes: State of the Art and Future Directions"; V4 No 4, pp387-404, Dec. 1999

[Briand99b] Briand V.R., Shull F, Lanubile F; "Building Knowledge through Families of Experiments"; IEEE Transactions on Software Engineering; V 25, No 4, pp 456-473, July 1999

[Briand01] Briand L.C, Bunse C, Dayle J.W.; "A Controlled Experiment for

Evaluating Quality Guidelines on the Maintainability of Object-Oriented Designs"; IEEE Transactions on Software Engineering, V 27 No 6, June 2001

[Briand01b] Briand L.C. Wust J; "Integrating Scenario-Based and Measurement-Based Software Product Assessment"; Journal of Systems and Software, V 59 No 1, p 3-22, October 2001

[Bruegge04] Bruegge B; et al. Object-Oriented Soft-ware Eng. 2nd Ed. Prentice Hall NJ; 2004

[Buchanan98] Buchanan J, Henig E, Henig M; "Objectivity and Subjectivity in the Decision Making Process" *Annals of Operations Research* 80 pp 333-345 1998.

[Chatz04] Chatzigeorgiou A, Xanthos S, Stephanides G; "Evaluating Object-Oriented Designs with Link Analysis"; Proc. 26th Int. Conf. on Software Engineering (ICSE); 2004

[Clegg94] Clegg C; "Psychology and Information Technology: The Study of Cognition in Organizations"; British J. of Psychology 85 449-477; 1994

[Cockburn96] Cockburn A; "The Interaction of Social Issues and Software Architecture"; Communications of the ACM; Vol.39 No.10 40-46, October; 1996

[Cockburn01] Cockburn A, Williams L; The Costs and Benefits of Pair Programming" Addison-Wesley Longman Publishing Co. Inc. Boston, MA, The XP Series, 2001

[Cockburn02] Cockburn, A; Agile Software Development, Addison Wesley, 2001

[Cohen93] Cohen MS. Three paradigms for viewing decision biases. In: Klein GA, Orasanu J, Calderwood R, Zsambok CE, editors. Decision making in action: models and methods. Norwood, NJ: Ablex Publishing Corporation; 1993.

[Conklin88] Conklin J, Begeman M; "gIBIS: A Hypertext Tool for Exploratory Policy Discussion"; ACM Trans. Office Information Systems, V6 No.4 303-331 October; 1988

[Cooksey95] Cooksey R.W. Judgment Analysis: Theory, Methods and Applications; Academic Press; 1995

[Cooper99] Cooper A; The Inmates are Running the Asylum 1st Ed.; Sams

Publishing; 1999

[Curtis88] Curtis B, Krasner H, Isco N; "A Field Study of the Software Design Process for Large Systems", Communications of the ACM, V.31 No.11, November; 1988

[Dawson03] Dawson, R, Bones Ph, Oates B.J, Brereton P, Azum M, Jackson M; "Empirical Methodologies in Software Engineering"; Proceedings of the Eleventh Annual International Workshop on Software Technology and Engineering Practice (STEP '03), pp 52-58, 2003

[DeMarco99] DeMarco T, Lister T; Peopleware, Productive Projects and Teams 2nd Edition; Dorset Publishing House, New York, NY; 1999.

[Dingsoyr02] Dingsoyr T; "Knowledge Management in Medium-Sized Software Consulting Companies"; Empirical Software Engineering, V 7 pp 383-386, 2002

[Dittrich07] Dittrich Y, … ; Understanding the Social Side of Software Engineering; A Special Issue in the Journal of Information and Software Technology, to appear Spring 2007

[Doherty93] Doherty, M; A Laboratory Scientist's View of Naturalistic Decision Making In: Klein GA, Orasanu J, Calderwood R, Zsambok CE, editors. Decision making in action: models and methods. Norwood, NJ: Ablex Publishing Corporation; 1993.

[Dromey95] Dromey R.G; "A Model for Software Product Quality"; IEE Transactions on Software Engineering; V 21 No 2; February 1995

[Dromey96] Dromey R.G; "Cornering the Chimera"; "IEEE Software; V 13, No 1, pp 33-43; January 1996

[Easterbrook06] Easterbrook S, Aranda J; "Case Studies for Software Engineers"; Proceeding of the 28th international conference on Software engineering; Tutorial Session, pp 1045-1046; Originally developed with Sim, S.E Perry D.E; ACM Press New York, NY, USA, 2006

[Emam97] Emam K; Hoeltte D; "Knowledge Management in Medium-Sized Software Consulting Companies"; Empirical Software Engineering, V 2 pp143-207 1997

[Erdogmus03] Erdogmus H, Favaro J; "Keep Your Options Open: Extreme Programming and Economics of Flexibility", in Extreme Programming Perspectives, M. Marchesi, G. Succi, D. Wells and L. Williams, Editors: Addison-Wesley, 2003

[Ericsson93] Ericsson, K., & Simon, H. Protocol Analysis: Verbal Reports as

Data, 2nd ed., Boston: MIT Press; 1993

[Fenton94] Fenton N, Pfleeger S.L; "Science and Substance: A Challenge to Software Engineers" IEEE Software, V 11 No 4, pp86-95 July 1994

[Fenton97] Fenton N, Pfleeger S.L; Software Metrics: A Rigorous & Practical Approach 2nd Ed; PWS Publishing Company; 1997

[Fisher89] Fisher G, McCall R; "JANUS: Integrating Hypertext with a Knowledge-based design Environment"; Proc. 2nd Annual ACM Conf. on Hypertext and Hypermedia; Pittsburgh, PA, 105-117; 1989

[Flanagan54] Flanagan, J. C. "The Critical Incident Technique"; Psychological Bulletin, 51(4), 327-358; 1954

[Flax06] Flax M, Fromin M, Jordan A, Nagl S, Radlegwski T, Smit T, Trabert O; "IBM Workplace Web Content Management for Portal 5.1 and IBM Workplace Web Content Management 2.5"; IBM Redbooks; http://www.redbooks.ibm.com/abstracts/sg246792.html?Open (Last Visited March 2007)

[Flyvberg01] Flyvberg B; Making Social Science Matter: Why Social Inquiry Fails and How it Can Succeed Again; Cambridge University Press 2001

[Fowler01] Fowler M; "Avoiding Repetition"; IEEE Software; January/February 2001

[Gamma95] Gamma E, Helm R, Johnson R, Vlissides J; Design Patterns: Elements of Reusable Object –Oriented Software; Addison-Wesley; 1995

[Gasson98] Gasson S; "Framing Design: A Social Process View of Information System Development"; Proc. Int. Conf. Information Systems, Helsinki Finland, 224-236; 1998

[Ghezzi91] Ghezzi C, Jazayeri M, Mandrioli D; Fundamentals of Software Engineering; Prentice Hall, Englewood Cliffs, NJ, USA 1991

[Gotterbarn01] Gotterbarn D; "Ethics in Qualitative Studies of Commercial Software Enterprises Ethical Analysis"; Empirical Software Engineering; V6 No 4 pp 301-304; December 2001

[Guindon90a] Guindon R; "Knowledge Exploited by Experts During Software System Design"; International Journal on Man-Machine Studies 33, 279-304; 1990

[Guindon90b] Guindon R; "Designing the Design Process: Exploiting Opportunistic Thoughts" Human Computer Interaction 5, 305-344; 1990

[Herbsleb01] Herbsleb J.D; Finbolt T, Grinter R.E; "An Empirical Study of Global Software Development: Distance and Speed"; Proceedings of the 23rd International Conference on Software Engineering, pp 81-90, Toronto, Ontario, Canada, 2001

[Herbsleb03] Herbsleb J, Mockus A; "Formulation and Preliminary Test of an Empirical Theory of Coordination in Software Engineering"; Proc. 9th European Software Engineering Conf./ACM SIGSOFT Symposium. on the Foundations of Software Engineering. ESEC/FSE; Sept 1-5, 2003;

[Highsmith99] Highsmith J; "e-Business Application Delivery: Managing Distributed Project Teams"; Cutter Consortium August 1999

[Highsmith02] Highsmith J; Agile Software Development Ecosystems; Addison-Wesley Indianapolis IN, 2003;

[Highsmith04] Highsmith J; Agile Project Management; Addison-Wesley, Boston MA; 2004

[Horvath99] Horvath L, Rudas I.J, Bito J.F; "Modeling Design Intent in Concurrent Engineering"; Industrial Electronics Society, IECON 99, The 25th Annual Conference of the IEEE, V 2 No 29, pp 968-972, 1999

[Horvath04] Horvath I; "A Treatise on Order in Engineering Design Research"; Research in Engineering Design; V 15 No 3 pp 155-181, 2004

[Hutchins95] Hutchins E; *Cognition in the Wild*, MIT Press, Cam. MA, 1995

[Iivari96] Iivari J; "Why Are Case Tools Not Used?" Communications of the ACM; V 39 No 10 October 1996

[Janis77] Janis I, Mann L; Decision Making: A Psychological Analysis of Conflict, Choice & Commitment; The Free Press New York, NY; 1977

[Jarczyk92] Jarczyk A.P.J, Loffler P, Shipman F.M. III; "Design Rationale for Software Engineering: A Survey"; Proc. 25th Hawaii Int. Conf. System Sciences, 2, 577-586, January; 1992

[Johnson05] Johnson-Laird, P.N. Mental models in thought. (2005) In Holyoak, K. and Sternberg, R.J. (Eds.) The Cambridge Handbook of Thinking and Reasoning. Cambridge: Cambridge University Press. Pp. 179-212.

[Johnson05b] Johnson-Laird P.N; "Mental Models, Sentential Reasoning, and Illusory Inferences"; in Held, C, Knauff M, Vosgerau G; Mental Models in Cognitive Psychology, Neuroscience, and Philosophy of Mind; New York, Elsevier 2005 ,

[Juristo01] Juristo N, Moreno A.M; Basics of Software Engineering Experimentation; Kluwer Academic Publishers, Boston MA; 2001

[Kahnema79] Kahneman D, Tversky A; "Prospect Theory: An Analysis of Decision under Risk"; Econometrica V47 No.2 263-292; 1979

[Kahnema82] Kahneman D, Tversky A; "Subjective Probability: A Judgment of Representativeness"; In: Kahneman D, Slovic P, Tversky A editors. Judgment Under Uncertainty - Heuristics and Biases Cambridge University Press; 1982

[Kaner96] Kaner S. with Lind L, Toldi C, Fisk S, Berger D; Facilitator's Guide to Participatory Decision Making; Sam Kaner, BC; 1996

[Kant84] Kant E, Newell A; "Problem Solving Techniques for the Design of Algorithms"; Information Processing & Management; V.20 No.1 97-118; 1984

[Karahasanovic05] Karahasanovic A, Anda B, Arisholm E, Hove S, Jorgensen M, Sjoberg D, Welland R; "Collecting Feedback During Software Engineering Experiments"; Empirical Software Engineering 10 (2) pp 113-147

[Karsenty96] Karsenty, L; "An Empirical Evaluation of Design Rational Documents"; Proc. of SIGCHI Conf. on Human Factors in Computing Systems; 150-156 Vancouver, BC; 1996

[Keeny93] Keeney R.L, Raiffa H; Decisions with Multiple Objectives: Preferences and Value Tradeoffs. Cambridge University Press; 1993

[Kemerer97] Kemerer C.F, Slaughter S; "Methodologies for Performing Empirical Studies: Report from the International Workshop on Empirical Studies of Software Maintenance"; Empirical Software Engineering 2 pp109-118 1997

[Kim92] Kim, J, Lerch F.J; "Towards a Model of Cognitive Process in

Logical Design: Comparing Object-Oriented and Traditional Functional Decomposition Software Methodologies"; Proc. of the SIGCHI Conf. on Human Factors in Computing Systems, 489-498, Monterey, California; 1992

[Kitchenham02] Kitchenham, B et al., J; "Preliminary Guidelines for Empirical Research in Software Engineering"; *IEEE Trans. Software Eng*, V.28 No.8: 721-734, 2002.

[Kitchenham04] Kitchenham B, Dyba T, Jorgensen M; "Evidence-Based Software Engineering"; Proceedings of the 26th International Conference on Software Engineering"; pp273-281; 2004

[Kitchenham95] Kitchenham B, Pfleeger S.L; "Towards a Framework for Software Measurement Validation"; IEEE Transactions on Software Engineering; V 21, No12, pp929-944, Dec 1995;

[Klein89] Klein G; "Recognition Primed Decisions"; Advances in Man-Machine Systems Research 5 47-92; 1989

[Klein89] Klein G et al; "Critical Decision Method for Eliciting Knowledge" IEEE Trans. Sys, Man and Cyber.; 19, 3, 1989

[Klein93] Klein M; "Capturing Design Rationale in Concurrent Engineering Teams", IEEE Computer, V26 No.1, 93-47, January; 1993

[Klein93b] Klein GA, Orasanu J, Calderwood R, Zsambok CE, editors. Decision making in action: models and methods. Norwood, NJ: Ablex Publishing Corporation; 1993.

[Klein98] Klein G; Sources of Power, MIT Press Cambridge, MA; 1998

[Kripend80] Krippendorff; Content Analysis; V5 Sage Pub. London 1980

[Laitenberger02] Laitenberger O, Bell T, Schwin T; "An Industrial Case Study to Examine a Non-Traditional Inspection Implementation for Requirements Specifications"; Empirical Software Engineering, V 7 No 4, pp345-374, December 2002

[Landry92] Landry M, Banvill C; A Disciplined Methodological Pluralism for MIS Research. Accounting, Management & Information Technology, 2(2), 77-97; 1992

[Lasher06] Lasher W, Hedges P, Fegarty T, Practical Financial Management 1st Canadian Edition; Nelson, Toronto, ON, CAN, 2006

[Lea93] Lea, D; "Christopher Alexander: An Introduction for Object-Oriented Designers"; ACM SIGSOFT Software Engineering Notes; V19, No 1, pp39-46, January 1993

[Lee91] Lee A.S; "Integrating Positivist and Interpretive Approaches to Organizational Research" *Organization Science* v 2 No 4 November 1991

[Lee96] Lee J, Li K-Y; "What's in Design Rationale?"; In: Moran T.P, Carroll J.M, editors. Design Rationale: Concepts, Techniques, and Use; Lawrence Erlbaum Associates, Mahwah NJ; 1996

[Lethbridge05] Lethbridge T.C, Sim S; Singer J; "Studying Software Engineers: Data Collection Techniques for Software Field Studies"; Empirical Software Engineering; V 10 No 3 pp 311-341, July 2005

[Lipshitz93] Lipshitz R; "Decision Making as Argument-Driven Action"; In: Klein GA, Orasanu J, Calderwood R, Zsambok CE, editors. Decision making in action: models and methods. Norwood, NJ: Ablex Publishing Corporation; 1993.

[Lipshitz93b] Lipshitz R; "Converging Themes in the Study of Decision Making in Realistic Settings"; In: Klein GA, Orasanu J, Calderwood R, Zsambok CE, editors. Decision making in action: models and methods. Norwood, NJ: Ablex Publishing Corporation; 1993.

[Lewin51] Lewin, K. (1951). *Field Theory in Social Science*. New York: Harper and Row.

[Lowgren95] Lowgren J; "Applying Design Methodology to Software Development"; Symposium on Designing Interactive Systems, Proceedings of the Conference on Designing Interactive Systems: Processes, Practices, Methods & Techniques; Ann Arbour, Michigan, USA, pp87-95; 1995

[Luce58] Luce D, Raiffa H; Games and Decisions: Introduction and Critical Survey, John Wiley & Sons Inc. New York, 1958

[MacLean96] MacLean A, Young R.M, Bellotti V.M.E, Moran T.P; "Questions, Options, Criteria: Elements of Design Space Analysis"; In: Moran T.P, Carroll J.M, editors. Design Rationale: Concepts, Techniques, and Use; Lawrence Erlbaum Associates, Mahwah NJ; 1996

[McDonald 98] McDonald D, Ackerman M; "Just Talk to Me: A Field Study of Expertise Location"; Proceedings of the 1998 ACM Conference on Computer Supported Cooperative Work; Nov 14-18, 1998

[McDonald05] McDomald J; "The Impact of Project Planning Team Experience on

Software Project Cost Estimates" Empirical Software Engineering V10 No 2, pp 219-234, April 2005

[McConnell98] McConnell S; "The Art, Science and Engineering of Software Development"; IEEE Software; V15, No 1, January 1998

[McGrath95] McGrath J; "Methodology Matters: Doing Research in the Behavioral and Social Sciences"; Human-Computer Interaction: Toward the Year 2000; pp 152-169, 1995

[Mak04] Mak J, Choy C, Lun D; "Precise Modeling of Design Patterns in UML"; Proc. 26th Int. Conf. on Software Engineering (ICSE); 2004

[Malhotra80] Malhotra A, Thomas J.C, Carroll J.M, Miller L.A; "Cognitive Processes in Design"; Int. J. Man-Machine Studies 12 119-140; 1980

[Melnik06] Melnik G, Maurer F: Comparative Analysis of Job Satisfaction in Agile and Non-Agile Software Development Teams, Proc. 7th International Conference on eXtreme Programming and Agile Processes in Software Engineering; June 2006.

[MerriamWebster07] Merriam-Webster Online http://www.m-w.com/ Last Visited May 2007

[Mingers01] Mingers J; "Combining IS Research Methods: Towards a Pluralist Methodology"; Information Systems Research, V 12 No 3, p 240-259, September 2001

[Mintzberg92] Mintzberg H, Quinn J.B; The Strategy Process; Englewood Cliffs, NJ; Prentice Hall 1992

[Montgom93] Montgomery H; "The Search for a Dominance Structure in Decision Making: Examining the Evidence"; In: Klein GA, Orasanu J, Calderwood R, Zsambok CE, editors. Decision making in action: models and methods. Norwood, NJ: Ablex Publishing Corporation; 1993

[Norman88] Norman D.A; The Design of Everyday Things; DoubleDay; New York; 1988

[Norman93] Norman D.A.; Things that Make us Smart: Defending Human Attributes in the Age of the Machine; Addison-Wesley; 1993

[Norman04] Norman D.A; Emotional Design: Why We Love (or Hate) Everyday Things; Basic Books New York, NY; 2004

[Olson96] Olson G.M, Olson J.S, Storrosten M, Carter M, Herbsleb J, Rueter

H; “The Structure of Activity During Design Meetings”; In: Moran T.P, Carroll J.M, editors. Design Rationale: Concepts, Techniques, and Use; Lawrence Erlbaum Associates, Mahwah NJ; 1996

[Orasanu93] Orasanu J, Connolly T; “The Reinvention of Decision Making”; In: Klein GA, Orasanu J, Calderwood R, Zsambok CE, editors. Decision making in action: models and methods. Norwood, NJ: Ablex Publishing Corporation; 1993

[Orlikowski91] Orlikowski W, Baroudi J; “Studying Information Technology in Organizations: research Approaches and Assumptions” Information Systems Research 2 1 1991

[Parnas86] Parnas D.L; Clements P.C; “A Rational Design Process: How and Why to Fake It”; IEEE Transactions on Software Engineering; V 12 No 2, pp 251-257, February 1986

[Patton02] Patton M.Q; Qualitative Research & Evaluation Methods 3rd Ed.; Sage Publications, California; 2002

[Penning93] Pennington N, Hastie R; “A Theory of Explanation-Based Decision Making”; In: Klein GA, Orasanu J, Calderwood R, Zsambok CE, editors. Decision making in action: models and methods. Norwood, NJ: Ablex Publishing Corporation; 1993.

[Perry94] Perry D, Staudenmayer N, Votta L; “People, Organizations and Process Improvement”; IEEE Software V.11 No.4, 36-45 July; 1994

[Perry00] Perry D; et al; “Empirical Studies of Software Engineering: A Roadmap”; Int. Conf. on S/W Engineering; Proc. of the Conf. on the Future of S/W Engineering; pp 245-255, 2000.

[Pfleeger97] Pfleeger S.L; “Status Report on Software Measurement”; IEEE Software, V14, No 2, pp 33-43; March 1997

[Pfleeger05] Pfleeger S.L; “Soup or Art? The Role of Evidential Force in Empirical Software Engineering”; *IEEE Software* Jan-Feb 2005 V 22 No. 1 pp. 66-73, 2005.

[Phillips84] Phillips LD; “A Theory of Requisite Decision Models”; Acta Psychologica 56 29-48; 1984

[Polanyi66] Polanyi, M. "The Tacit Dimension". First published Doubleday & Co, 1966

[Poppend03] Poppendieck, M, Poppendieck T; Lean Software Development: An Agile Toolkit; Addison Wesley, Upper Saddle River, NJ; 2003

[Potts88] Potts C, Bruns G; "Recording the Reasons for Design Decisions" Proc. 10th Int. Conf. on Software Engineering; 418 – 427, Singapore; 1988

[Rasmus93] Rasmussen J; "Deciding and Doing: Decision Making in Natural Contexts"; In: Klein GA, Orasanu J, Calderwood R, Zsambok CE, editors. Decision making in action: models and methods. Norwood, NJ: Ablex Publishing Corporation; 1993.

[Rittel73] Rittel H; Webber M; "Dilemmas in a General Theory of Planning"; Policy Sciences 4, 155-169; 1973

[Robillard98] Robillard P, d'Astous P, Detienne F, Visser W; "An Empirical Method Based on Protocol Analysis to Analyze Technical Review Meetings"; Proceedings of the 1998 Conference of the Centre for Advanced Studies on Collaborative Research; p20, Toronto, Ontario, Canada; 1998

[Robillard98b] Robillard P, d'Astous P, Detienne F, Visser W; "Measuring Cognitive Activities in Software Engineering"; Proceedings of the 20th International Conference on Software Engineering pp 292-299, 1998

[Royce90] Royce W; "Managing the Development of Large Software Systems"; Proceedings of the 9th International Conference on Software Engineering; Monterey, California, pp 328-338; USA, 1987

[Rugaber90] Rugaber S, Ornburn S, LeBlanc R; "Recognizing Design Decisions in Programs" IEEE Software; 1990

[Ruhe04] Ruhe, G; "Software Engineering Decision Support – Methodology and Applications" Innovations in Decision Support Systems, ed. Tonfoni and Jain, International Series on Advanced Intelligence, V 3, 2003, pp 143-174

[Saaty80] Saaty T.L; The Analytical Hierarchy Process: Planning, Priority Setting, Resource Allocation, McGraw-Hill, New York; 1980

[Schach99] Schach S.R; Classical and Object-Oriented Software Engineering, with UML and Java; Fourth Edition, WCB McGraw-Hill, 1999

[Schwandt01] Schwandt01 T.A; Dictionary of Qualitative Inquiry; Sage Publications Inc, California, USA, 2001

[Seaman97] Seaman C, Basili V; "The Study of Software Maintenance

Organizations and Processes"; Empirical Software Engineering, V 2 No 2, pp 197-201 1997

[Seaman99] Seaman C; "Qualitative Methods in Empirical Studies of Software Engineering"; IEEE Transactions on Software Engineering; V 25 No 4; pp557-572, July 1999

[Seaman01] Seaman C; "Ethics in Qualitative Studies of Commercial Software Enterprises: Case Description"; Empirical Software Engineering; V 6 No 4; pp299-300, December 2001

[Segal05] Segal J, Grinyer A, Sharp H; "The Type of Evidence Produced by Empirical Software Engineers"; International Conference on Software Engineering, Proceedings of the 2005 Workshop on Realising Evidence-Based Software Engineering; St. Louis, Missouri, p1-4; 2005

[Sharp91] Sharp, H.C. 'The Role of Domain Knowledge in Software Design', in *Behaviour and Information Technology*, **10**(5), 383–401, 1991

[Shum96] Shum S.B; "Analyzing the Usability of a Design Rationale Notation"; In: Moran T.P, Carroll J.M, editors. Design Rationale: Concepts, Techniques, and Use; Lawrence Erlbaum Associates, Mahwah NJ; 1996

[Siddall72] Siddal J; Analytical Decision-Making in Engineering Design; Prentice Hall Inc., Englewood Cliffs NJ; 1972

[Simon55] Simon H; "A Behavioural Model of Rational Choice"; Quarterly Journal of Economics; V69 Issue 1 99-118; 1955

[Simon73] Simon H; "The Structure of Ill Structured Problems"; Artificial Intelligence 4, 181-201; 1973

[Simon96] Simon H; The Sciences of the Artificial; MIT Press, Cambridge MA, 3rd Edition

[Sonnetag98] Sonnetag S; "Expertise in Professional Software Design: A Process Study"; Journal of Applied Psychology V.83 No. 5, 703-715, October; 1998

[Sounda04] Soundarajan N, Hallstrom J.O; "Responsibilities and Rewards: Specifying Design Patterns; Proc. of the 26th Int. Conf. on Software Engineering (ICSE); 2004

[Stapleton97] Stapleton J; DSDM: Dynamic System Development Method; Addison Wesley 1997

[Stemler06] Stemler, SE. “A Comparison of Consensus, Consistency and Measurement Approaches to Estimating Interrater Reliability” Practical Assessment, Research & Evaluation, 9(4). Retrieved October 30, 2006 from http://PAREonline.net/getvn.asp?v=9&n=4.

[Stirling03] Stirling W.C; Satisficing Games and Decision Making, with Applications to Engineering and Computer Science; Cambridge University Press, Cambridge, UK; 2003

[Sullivan99] Sullivan K, Chalasani P, Jha S, Sazawal V; “Software Design as an Investment Activity: A Real Options Perspective” Real Options and Business Strategy: Applications to Decision making, L Trigeorgis ed (London, England: Risk Books), pp215-261, 1999.

[Taylor07] http://www.quality.org/TQM-MSI/taylor.html (Last visited March 2007)

[Tichy98] Tichy, W; “Should Computer Scientists Experiment More”; IEEE Computer V31, No.5 pp 32-40, May 1998.

[Tversky72] Tversky A; “Elimination by Aspects: A Theory of Choice”; Psychological Review; V.79 No.4, 281-299; 1972

[Tversky82] Tversky A, Kahneman D; “Judgment Under Uncertainty – Heuristics and Biases”; In: Kahneman D, Slovic P, Tversky A editors. Judgment Under Uncertainty - Heuristics and Biases Cambridge University Press; 1982

[Tversky82b] Tversky A, Kahneman D; “Belief in the Law of Small Numbers”; In: Kahneman D, Slovic P, Tversky A editors. Judgment Under Uncertainty - Heuristics and Biases Cambridge University Press; 1982

[UML07] Unified Modeling Language; http://www.uml.org; Last Visited March 2007

[Vicente03] Vicente K; The Human Factor; Alfred A Knopf Canada; 2003

[Walker03] Walker R.J et al.; “Panel: Empirical Validation – What, Why, When and How”; Proc. 25th Int. Conf. on S/W Engineering; pp 721-722, 2003.

[Walz93] Walz D.B, Elam J.J, Curtis B; “Inside a Software Design Team: Knowledge Acquisition, Sharing and Integration”; Communications of the ACM, V.36 No.10 October; 1993

anni[Wardell91] Wardell R.W.; O'Grady J; "The Practice of Ergonomics in Industrial Design"; 24th Annual Conference of the Human Factors Association of Canada 1991

[Websphere07] http://www-306.ibm.com/software/websphere/ (Last Visited April 2007)

[Weinberg98] Weinberg G.M; The Psychology of Computer Programming; Dorset House Publishing, New York; 1998

[Williams03] Williams L; "Agile Software Development: It's About Feedback and Change"; IEEE Computer, V.36, No.6, 39-43 June; 2003

[Wholin04] Wholin C; "Are Individual Differences in Software Development Performance Possible to Capture Using a Quantitative Survey"; Empirical Software Engineering; V 9 No 3 pp211-228, September 2004

[Yin02] Yin R.K; Case Study Research: Design and Methods, 3/e Thousand Oaks, CA: Sage Publications, 2002

[Zannier05] Zannier C, Maurer F; "A Qualitative Empirical Evaluation of Design Decisions"; International Conference on Software Engineering; Proceedings of the 2005 Workshop on Human and Social Factors of Software Engineering, St. Louis, Missouri, p 1-7, 2005

[Zannier07] Zannier C, Maurer F; "Comparing Decision Making in Agile and Non-Agile Software Organizations"; Proceedings of the 8th International Conference on Agile Processes in Software Engineering and Extreme Programming; Como, Italy, Springer 2007

[Zannier07b] Zannier C, Chiasson M, Maurer F: A Model of Design Decision Making based on Empirical Results of Interviews with Software Designers , Understanding the Social Side of Software Engineering Qualitative Software Engineering Research, A Special Issue of the Journal of Information and Software Technology, to appear Spring 2007

APPENDIX A - CASE STUDY SUMMARIES

Interview Participant #1: CS1.1

This mentor continuously compared what he saw people doing when performing a design change with what he did when performing a design change. Design decisions should reflect an understanding of the design and every element of a design should capture some important concept in the domain. Cues that a design change needs to be made came from new features or recognizing that the implementation of a new feature is much more difficult than it "out to be". Many teams think that the only productivity is in developing new features rather than changing existing design to better capture the business need. The knowledge to accomplish good design comes from working with good people and with good code. People do not look up knowledge in a book, "at least not when they're expected to, books provide you with an awareness of design." Similarly the use of past experience and the memory of a general pattern of a solution in a new problem happens all the time. When asked about options, this mentor said people settle on the first idea they have. However, brainstorming is a fast process an designers should brainstorm numerous ideas, use their "speech centre" to experiment with teams and get into that place where ideas are just "popping out on variation." Time pressure restrains people's ability to make a decision which is a terrible mistake, "because that half hour in a room will save you hours at the keyboard. There's a very strong tendency to take as short a path as you can to a working feature and just keep going, especially when you're under time pressure."

Interview Participant 2: CS1.2

This mentor discussed group decision making and compared what he saw people doing and what he thought people should be doing. Often decisions are made based on power, but groups should get down to the criteria for a decision, why a criterion is important to an individual involved in the decision making. Cues that a design change needs to occur come from feedback mechanisms built into the software process: integration times and velocity, in agile terms, design documents that are simple or aesthetically pleasing, in non-agile terms. This mentor said that while a lot of knowledge transfer comes from working with other people, there is an openness, a curiosity or a "thrust for learning" in certain people who actively chase knowledge. Past experience is somewhat useful but "just because something succeeded on one project does not necessarily mean it's the right solution on the next project. Every project is different in part because every project has different external goals and these multiple external goals should be the criteria on which people make decisions. Options should be kept open as late as possible into the project, not 50 options, but 3 or 4 options, because 'you can always accumulate more information up to that point. Everybody looks at options but they don't keep them open long enough." Designers should respond to time pressure but not let time pressure force them into a bad decision.

Interview Participant #3: CS1.3

This developer discussed a specific design change he championed. He was not happy with the existing design, talked to friends about a possible solution, read up on the solution and sold the idea to the business managers. The cues came from his own dealings with the

system and “feeling the pain” because he was “struggling with the HTML library.” He heard some frustrations from others and through the code was verbose, awkward and difficult to navigate. He had some experience with XML and XSLT in the past but at that time it was too immature to use. He bought a book on XSLT and XML to see if it would be up to the task, and he read the book and tried out the solution. HE also had previous experience in the problem of refactoring but said that was a pretty generic re-use of past experience. He “thought a bit about to the design approaches but mostly decided pretty quickly that this solution was a good way to go.” He was not pressured by time and the only external goal was a desire to try something new.

Interview Participant #4: CS1.4

This developer discussed a new test framework he developed with his colleagues in their spare time. They had to “re-think the way a framework worked based on how you would do things in .NET or C#”. The cues came from feedback from developers requesting a new framework and recognizing the capabilities of .NET and C3. At the time there were not a lot of books to read on the subject, mostly this developer just thought of a better way of doing things. He also could not rely on past experience because it was something new. They looked at one other solution to the problem using a metafile and talked about using XML, but wanted to maintain something similar to the JUnit experience. There was not a lot of time pressure because the task was ancillary to their jobs, and the main external goal was to develop a deeper understanding of .NET.

Interview Participant #5: CS1.5

This mentor discussed his experiences when changing design, from an abstract level. Design change is “making little tiny, atomic changes that are sort of making the design better globally.” Cues that a design change needs to occur come from communication and finding better words to express what you want. There are people who buy books and people who don’t and “the people who don’t, unless they have their noses pushed into some other learning trough, just pretty much stopped learning.” This mentor also knows a lot of stuff having done it and having discovered it. Stories about experiences are a very compelling way of communicating but mostly for this mentor, “everything I ever learned is just in there… I have no idea where it came from.” When considering options this mentor presented two sides. “If a person is not very experienced in whatever they are working in, I think they are well off to pick a simple solution because it is in building the bridge between one side of the river and the other that we learn most about getting across the river.” This mentor also said he always thinks about alternatives when programming. Time pressure is totally destructive and causes flaky decision making External goals do exist but there are ways around external goals, by representing the goal in terms of the project to make the customer realize this goal is something s/he wants.

Interview Participant #6: CS1.6

This mentor discussed what he coached software development teams to do and how it related to design decision making. He used specific examples periodically. The decision

about when a team is “done” with design has no set metric or test other than to code up a design, see if it works, and satisfy the need to for documentation. Cues that a design change needs to occur comes from the comfort level of the team to proceed with the design as it is. The significant knowledge used is the knowledge people bring into the room with them, a willingness to use available information tools and who they talk to. “There’s a trigger that says do you understand the story enough to do it and if not what else do we need?” The idea of task estimation encourages people to use past experiences on a new project. Tradeoffs are evaluated and options are weighed most formally in the iteration planning meetings, but he does not coach to do a real exhaustive analysis of the options because it would just take to long.

Interview Participant #7: CS1.7

This developer discussed design change from a high level of all of his experiences. Design changes are made to make the design better than it was before and a good design is a testable design. Cues that a design change need to be made come from a feature that is not in the system that needs to be in the system. It is “exceedingly rare that I would notice a refactoring is required if I’m not in the process of adding a feature or fixing a problem.” Knowledge used in design decisions comes from “learning simple rules, following them to gain some benefit and then seeing how far I can go just following those simple rules.” These simple rules tend to be somewhat universal, they are widely applicable, so the use of past experiences in new situations happens “quite a bit”. This developer said he absolutely tries out different options in a design problem, when he isn’t sure where the problem is going. However, often, “if I’m quite confident I know a good solution that does what I need, I will just do that, and that’s a question of experience.” The more time pressure there is, the less patient he is in waiting for a bad idea to show itself as a bad idea, and there is a tendency to conclude too quickly that an idea is not bad. The main external goal for this developer is practicing and experimenting with scenarios.

Interview Participant #8: CS1.8

This developer discussed a specific design change he championed. He was required to re-design the way a fee was handled in the existing application. The cues that the design change needed to occur was that, “when a customer comes in and says ‘we need a new fee based on this’, it’s a substantial amount of work to implement if its not something already supported in the system.” The knowledge used in the design change came from discussions with other developers and some searching on the internet. Some experiences from school were used but, “a lot of other things I had to look up just because I didn’t remember them.” The design solution he implemented was pretty close to the first idea he and another developer came up with and after prototyping it and seeing that it worked they didn’t’ consider too many other ideas. The only external goal he had was that the application had to work with Java.

Interview Participant #9: CS1.9

This mentor compared what he saw developers doing and what he though should be done, around the topic of design change. He tries to get developers to code and design with short time horizons and to sop and think about design more. Cues for a design change come from the code and not as much from documentation, although there is some value in some documentation. This mentor isn't sure if developers have to make a design change, and isn't sure if the changes developers make always improve the design. Past experience is definitely re-used. "A programmer finds one way to solve a problem and that is the way he is going to solve that problem for the rest of his career." In considering options to design change most people do well in considering options on the big things but many designers or architects let other factors colour their decision. He tries very hard not to put a team under pressure, but finds sometimes it factors in although it doesn't necessarily hurt the decision. "So I don't think the design issues are really affected by time too much." External goals do affect design decisions, even though they're orthogonal to what the real goal is.

Interview Participant #10: CS1.10

This developer discussed design change by discussing a few concrete experiences he had in changing the design. He described a project where he had to integrate his company's application with another application that a shared customer was using and there were a few design changes within that problem. Cues that a design change needed to occur came from feedback from the customer that wasn't entirely clear, the 3rd party company and some general invention on his part. The knowledge used in the problem came from design patterns books and ideas he had on his own. Concrete past experiences weren't used but general experience with the programming languages was used. The consideration of options varied depending on the problem. Once he found a design pattern that suited his problem he didn't look at other options. In a decision to use COM as an underlying communication protocol he did not consider other options because it was too simple to just use COM. However in the design of the high level communication protocol with the 3rd party software they did consider options and prototype before committing to one solution. He was not under time pressure but he did have pressure from the marketing department of the 3rd party software company to use XML and get good press in the use of XML.

Interview Participant #11: CS1.11

This mentor talked about design change as constant change and discussed the management of putting a new product on the market. Design starts with an idea for the general market, people in marketing refine the idea to start selling it and experts constantly refine what they're doing with the design. Cues come from ideas. In one situation this mentor described a talk she gave leading to someone's new idea and ultimately product development. "New products are not planned, they happen." Knowledge used in new product development comes from general education and general knowledge and experience is re-used in a general sense, not a specific re-use of experience. How many options are considered is a function of how fast you have to get to market. "When you bring a product to market you don't do a lot of options. You want to have something in the market that proves the concept." All goals should be driven from a financial model and the financial model should

be a viable one. Time pressure is a negotiation between quality and marketing and time pressure is good. If you don't have some time pressure you should create some.

Interview Participant #12: CS1.12

This mentor discussed a design change he had coached. Design change was done incrementally, the team communicated about the design on a high level then sub-teams focused on details. Cues that a design change needs to occur come from customer need. The knowledge used in the design change comes from educational background and general knowledge of a technology. Similarly experience was used in a general way and you have a bag of tricks that you use. Options are considered, when you look at 3 things and say, "I have no idea which is better let's just do an experiment and find out where it goes." Once in a while considering options seems to be worth it. In other situations you hone in on areas you're familiar with. External goals usually come up early Time pressure does impact decisions and usually the ganger is too much time pressure rather than too little.

Interview Participant #13: CS1.13

This developer discussed a specific design change he had made when working on a side project on his own time. He got a piece of code working then his decision was to refactor the code to separate business logic from user interface code to make the code easier to test. The cue that a design change needed to occur was recognizing the code was long and messy and seeing that it did not align with a best practice of separating business logic and presentation code. The knowledge for this change came from working with colleagues, working on other projects and applying best practices, not from textbooks. Similarly, experience was used in the generic sense, no specific experience was re-used. This developer did not consider any options. The goals were to separate business from presentation code and to make testing easier. There were no other external goals. There was no time pressure on the project.

Interview Participant #14: CS1.14

This developer discussed a design change to a legacy system to make one component better known to the development team. The cues that a design change needed to be made came from developer complains and from this developer recognizing the code was not properly done. The Knowledge for the decision came from always working in software development and from working with really good developers. He said he never uses textbooks. Past experiences are always re-used. For the particular problem he described he did not consider any alternatives because it wasn't a difficult decision. He said that usually he spends several hours on major decisions. The external goal was to produce a complete set of tests but they needed to provide something to customers and that was the overriding criterion.

Interview Participant #15: CS1.15
This developer discussed a design change where he was required to move all the caching in a project into one large cache. The cues that this design change needed to be made was hearing feedback from the development team or the difficulties he had with adding new code because of problems in the existing system. The knowledge used in the solution was a design pattern what they published. Design patterns, "are just engrained and you just see things in a certain way." Past experience he used was specific to design patterns. They had one big discussion about another alternative but mostly they didn't have any other reasonable alternative. An external goal was to learn or teach Java and there was a lot of time pressure but he didn't feel it because they were making progress.

Interview Participant #16: CS1.16
This developer discussed a very specific change made to the backend of an existing application. The TSX changed a 3rd party application which prompted design changes at the code level. Cues that a design change needed to be made came from recognizing a method was missing. No textbooks were used except reference books that were shared by the development team. Core programming skills and system design skills were based on past experiences but the customer domain was not based on past experience. Small options were considered, like introducing one large method, or many small methods, this developer said somewhat sarcastically, but ultimately, there was someone in charge who had to approve everything. There was significant time pressure, unrealistic deadlines, set by someone else.

Interview Participant #17: CS1.17
This developer discussed a decision to change to a new design paradigm from an old design paradigm which involved refactoring most of the code. The cue that a design change needed to occur was that the presentation logic was mixed with the business logic so the code was difficult to test and to read. The knowledge used to make the decision was a specific paper about why a template design is good for separating presentation logic from business logic and a book on J2EE patterns. Past experience was used significantly. This developer had done a lot of web based systems and worked on projects with similar types of problems and had used a similar design solution when he had the same problem previously. Limited options were considered, it was more important to pick something and follow through. A proof of concept was implemented. There were few external goals, only determining if the code looked ok. There was not significant time pressure, just an expectation that the developer would make some progress.

Interview Participant #18: CS1.18
The developer discussed multiple design changes he had experienced and said that all changes are predicated by cost and market. Cues that a design change needs to occur comes from feedback from resellers of the software product, about customer requests. This often occurs at tradeshows. Knowledge used in the design change comes from a general use of past experiences and a general use of attending Microsoft conferences. Thus, past experiences are re-used often in specific and general ways. Options are considered

depending on the type of problem. Some things are not an option because, “we’re a Microsoft house and we’ve got people in the office that are certified on sequel server, so there is not real option between Oracle and DB2 and sequel server.” On other design decisions, they weight options, when there are options that seem viable. An example was determining how to do credit card processing. They did a case study of the options and reviewed the options as a group. The main external goal is a desire to do a million dollars worth of sales.

Interview Participant #19: CS1.19

This developer spoke about design changes from a testing perspective. Poor workflow and inconsistencies in an existing HTML legacy system prompted small changes to the system but complexity of testing prompted the decision not to make any large changes all at once. Cues that a design change needed to be made came from bringing new people into the testing group and they could recognize problems in the system that other people had become used to. The knowledge used in the design change was mostly context specific and a general use of past experiences. They had options to implement in bigger stages but they didn’t have enough time to test the system. An external goal was to not overwhelm the customer with too many changes and listening to customer feedback.

Interview Participant #20: CS1.20

This developer discussed a design change he had championed. A request of a scripting engine led to recognizing code was dispersed across the system and could be put into one place despite the fact that it would make testing more difficult. Cues that a design change needed to occur came from a client request and a desire to make the design easier to manage in the future. This developer didn’t know where he got the idea to make the change other than that it was a basic encapsulation idea and he had learned that in the course of his education and experience. “I can’t think of a particular [past experience], I mean it just felt like the right thing to do. So possibly, subconsciously I’d done it before or seen it.” One of the other team members was arguing heavily for an option but it was in an effort to make testing easier. An external goal was to clean up the code and time pressure did not impact the decision because the project was on track.

Interview Participant #21: CS1.21

This developer discussed refactoring a user interface screen to make it easier for an end user to use. He made it easier to add agreements (an object in the business domain) to a user interface. The cues that a design change needed to occur was a customer request. The knowledge used in the design solution came from the “top of his head” This developer said he doesn’t read textbooks at all. He did re-use past experience in solving design change, in a generic sense. He did not consider any alternatives. The external goal was to make the system more user friendly, and he felt significant time pressure, he had a week to make the change.

Interview Participant #22: CS1.22
This developer discussed a design change in the architecture he championed where he moved logic from a middle tier to a stored procedure. It meant moving functionality from the client layer down to the database. Cues that the design change needed to occur came from technical and performance complaints from users of the system. The knowledge used came from best practices. "Sybase provides you with… online cds… just like a knowledge base I guess, as to how you would do things." He also re-used previous experiences that had similar performance problems. He, with others, did consider options for about a week before everyone agreed on the chosen option. External goals were enhancing maintainability and support for the application. He did not feel much time pressure on the design change.

Interview Participant #23: CS1.23
This developer discussed a design change he championed where he didn't think the project was worth doing, but "that was overruled so I set out to find a way to solve the problem in a way that would allow us to not waste our time doing it. To get some other goals out of it. … We have an existing product, it's a heavyweight desktop application which has its own proprietary data format, and … the client was looking for a way to access data from that and put it into word documents … they wanted to be able to pull data out, put it into letters and spreadsheets and things like that." The knowledge used in the problem came from the Gang of Four design patterns book and from working with very smart people. He also used an earlier project with a similar problem in this problem. He considered two options but did not prototype anything. There were no external goals but the project was under immense time pressure.

Interview Participant #24: CS1.24
This developer discussed a design change made to an existing application to handle a very large data import. The cues that a design change needed to occur was recognizing the need for synchronicity and a desire to provide automated feedback to the user of the system. The knowledge used in the design change, "came from the culmination of a bunch of things. One is I was recently reading some things on EJBs and asynchronicity." This developer also picked through design patterns. The re-use of experience was in a general sense only. He did not consider options on this design change and said, "I think there comes a point where you know something is going to be the right thing for the right task, the right solution for the task at hand." There were smaller external goals to have more control over the data and to be able to edit the data better. There was a "ridiculous" amount o f time pressure on this task. For this developer the task was, "one of the toughest 8 months I think I've ever worked in my career."

Interview Participant #25: CS1.25
This developer discussed two different design changes he made and compared the approaches he took to each. In one situation there were performance problems that had to

be detected and fixed. Cues that the change needed to be made came from complaints from the field an out of memory messages, knowledge used in the design change came from general experience and "accumulation" of lessons learned. Similarly experience was re-used in the general sense. There were not a lot of options considered because the problem was, "entrenched" in existing architecture. There was some time pressure on this problem and "no real other goals than just wanting to come up with the simplest, most maintainable solution". In another situation this developer was working on a "pet project" and considered plenty of options and compared and contrasted them. He "wanted to try different things and really kick the tires and be able to speak intelligently." There was not time pressure in this situation.

Observations – Company A, CS2.1A

Design problem: BUG FIX: Client had a problem over the weekend he had to fix.

Case Study Summary

A bug fix comes in and the designer searches on internet, using Edit-Find during search. He searches more on internet and compares what he found to his own stuff. Searches and reads more, in own system he does an Edit-Find. Tries to contact someone on the phone, unsuccessfully, searches internet. Takes some code from internet and uses it. Makes some changes to existing code, and after changes runs system in debug mode. Makes more changes and goes back to internet search. Goes back to code and continually changes, alternating between internet and code with emphasis on running system and changes. Reads solutions on the internet but mostly changes code until he gets it working

Observations – Company A, CS2.2A

Design Problem: BUG FIX record lost at customer's system (just one)

Design Problem: Whether or not to include the fix in next release.

Case Study Summary

Bug fix from a Tim Horton's, designer looks @ log file where error is and at code. Calls technical support staff member to ask about part of system. Runs through scenario of problem with this technical support staff member. Recognizes something in log file that doesn't make sense for the date. Compares this to a good version of the log file. Reads code. Tells technical support staff member problem could impact/halt next release. Calls a dealer, who is the contact for the customer, and they review problem together. Says needs approval from technical lead/manager to go ahead with fix. Business manager gives approval and tells him three places where record should go. Designer wants to put record in a place that fits initial framework. He thinks of another scenario that causes a problem and says he needs to talk to technical manager who knows inner workings of system. Technical manager makes suggestion for solution and designer explains why it wouldn't work.. Technical manager proposes another idea and designer says why that wouldn't work. They run some scenarios together. Debate if they should hold off on release. Technical manager says no because of time and that it is a third party company's problem anyway. Technical manager and designer talk about XML solution they had talked about before. Technical manager says to leave the problem for now and says there are business work-arounds. Designer brings up technical problem. They talk about XML solution again. Technical

manager says to work on something else instead, designer fixes problem for that one customer.

Observations – Company A, CS2.3A

Design Problem: FEATURE REQUEST: Figure out format of data going between cashier's terminal and back to kitchen monitor because customer wants a feature.

Case Study Summary

Feature request given higher priority than previously. Changes code, runs system, uses Edit-Find. Message box pops up twice, called once though, he says not important. Says log file tells him there's a problem. Changes code, looks at log file, talks aloud about model. Focuses on log file and runs in debug. From debug he says what he thinks was happening. Changes code and says what he thinks is happening. Puts in a LOC "Because its just good programming practice." Talks in general about good programming practices". Runs system and gets an error. Fixes it and says he's making progress. Changes code and says he saw problem in log file, related to problem. Changes code and says what he thinks will happen. Traces program. Goes to internet search. Copies code form internet and pastes it into his problem. Says why he thinks it wasn't working and so took code form internet. Changes code, runs system, "That didn't work either." Thinks, deletes code from internet and writes own. Says idea for code comes from what he needs to see in log file. Says internet code was useless. He tries to remember utility he wrote a while ago that he deleted. Changes code and says appears to be working but not sure. Changes code to get rid of useless stuff, remove clutter and because didn't make sense. Says worried log file will get big and wants short message that's useful. Changes code based on what's useful and what he knows about system. Changes code and compares 2 things in log file. Says log file now has enough stuff to be useful. Says it looks like its sending. Puts it into real environment. It works on his machine, not on real machine. Talks out loud about why not, running a scenario. Changes part of a system. Continues to change system. Continues to pick something to focus on. Continues to explain what he thinks is happening. Continues to test system. Compares 2 XML files and notices a difference. Explains why there's a difference. Continues to change system. Continues to pick something to focus on. Sometimes deemphasize or bolster. Continues to explain what he thinks is happening. Continues to test system.

Observations – Company A, CS2.4A

Design Problem: BUG FIX: Problem in system so goes through different versions of the code comparing.

Case Study Summary

Goes through different versions of code comparing them. Codes one line of code. Reads personal notes. Codes a few lines of code. Looks at personal notes. Calls someone in office to ask if piece of code is used. Looks at source code printout. Uses lookahead function of tool and changes some code. Copies and pastes some LOC from one file to another. Reads other piece of code for a while. Highlights entire section of code and copies and pastes it over. Changes code and uses lookahead function. Copies and pastes from old to new. Periodically does own coding and uses lookahead function. Copies and pastes ~10LOC.

Look at source code printout. Continues comparing 2 files, copies and pastes sometimes, codes sometimes.

Observations – Company A, CS2.5A
Design Problem: FEATURE REQUEST: Technical lead gives project to a developer – Customer wants to have coffee monitor feature .
Case Study Summary
Uses kitchen video (another project) to describe what he wants. Lists out 2 aspects of problem for developer to solve. Developer presents two options. Technical lead specifies technical criteria: do it in .NET. Technical lead gives developer constraint about not losing data and having header. Developer provides possible solution. Technical lead asks question with a scenario. Technical lead uses kitchen video again to explain what he means. Technical lead gives requirements about attributes of model. Developer asks follow-up question and technical lead makes a suggestion. Developer says its like another project they have worked on and technical lead agrees. Developer asks question and technical lead follows up with answer. Technical lead breaks down tasks of project for developer into 3 pieces. Technical lead tells developer to look at other example and tells him not to worry about one aspect of problem. Technical lead idea came from seeing a competitor's product and from a dealers' request.

Observations – Company A, CS2.6A
Design Problem: FEATURE REQUEST: Partner company wants part of the company's software to include keystrokes so that a user can switch from their software to the partner company's software easily.
Case Study Summary
Feature request from partner company, technical lead describes current situation of UI. Says won't make money off feature and that the feature is a pain to implement, but they kind of have to do it because the partner company got them into two very large customers.

Observations – Company A, CS2.7A
Design Problem: BUG FIX for one of their software products.
Case Study Summary
Technical lead looks at code buts its been a long time and never enough comments. Reads code, goes to Microsoft access database, switches between functions. Read through gave him an idea. He tests with some data, Finds a solution, just to offer user option to proceed. Doesn't remember why he didn't allow the option before.

Observations – Company A, CS2.8A
Design Problem: BUG FIX with ping
Case Study Summary
Reads source code but says "that was fruitless". Goes to test room to test system. Did a lot of trial and error, see what happened each time. Eliminate possibilities by running different

scenarios. Says it seems like a timing issue tests and says he's right. Goes back to desk looks through help file finds 4 types of pings. Tries the 2nd option because "it was next on the list". Gave some technical reasons for problems with options 3 & 4. Changes code, goes to test it, it didn't work. Re-checks stuff in help file and says that's what he's doing. "In an effort to save time, I'm going to do … a band-aid." Explains logic of his modeling idea and that he thinks it'll work. Changes back to original ping so no unknown problems are introduced. Tests it, says "even if its not perfect, its better than dying"

Observations – Company A, CS2.9A

Design Problem: FEATURE REQUEST: working on a timer for the coffee monitor, has to have one timer for all 6 lists in the coffee monitor

Case Study Summary

Runs application to set # rows and columns and show me. Writes some code, looks at personal notes, thinks. Uses lookahead function and thinks for a long time. Tells me his biggest concern is all 6 lists use 1 method. Writes out a possible solution on paper and changes code. Uses lookahead and Edit Find and he changes code. Writes some code and runs system. Uses debugger and runs system after looking at help file. Explains why he thinks its not working. Changes some code, points at screen, thinking. Copies and pastes some code, test it and it works. Explains why it worked. Explains limitations and says an error message is no big deal. Tests it, predicting when it will fail and he is right. Tells me how he's going to change it. Changes it and explains when it should work and it does.

Observations – Company A, CS2.10A

Design Problem: FEATURE REQUEST: has to set the system so times stop when the order is done and has to remove items from list.

Case Study Summary

He's done something like this before. He doesn't want to use a lot of memory. Writes notes on piece of paper. Runs system to test it out. Looks at code, thinking. Goes to help file quickly and to Google to find type that uses least memory. Changes a variable and says wants to use type that uses least memory. Copies and pastes code from one method to another. Makes a lot of changes in code and explains model to me. Changes code, runs with debugger. Says he is writing error handling but the error won't happen and says why. Tests system till he gets an error Changes code and says he thinks it'll work. It works. He explains why it worked.

Observations – Company A, CS2.11A

Design Problem: FEATURE REQUEST – do a replace function

Case Study Summary

Technical lead tells developer there are 2 ways to do it. They talk about format of message Developer says the model items he needs, technical lead doesn't agree. Developer asks to go through an example, technical lead explains problem to him using whiteboard. Technical lead tells him 2 different ways to do it. Technical lead explains each of the

options. Developer brings up a point, technical lead tells him its no big deal. Technical lead suggests one way is better and developer agrees very quickly.

Observations – Company A, CS2.12A

Design Problem: FEATURE REQUEST: has to make configuration form run once and only once.

Case Study Summary

Says he's done stuff like this before and copies and pastes and modifies code. Explains part of design to me. Shows me code he's copying and pasting and says why waste his time Changes code he's copied a bit. Adds in header from technical lead's requirements and tests system It doesn't work, changes code, says knows it isn't best thing to do.

Observations – Company B, CS2.1B

Design Problem: FEATURE REQUEST: Wants to incorporate more workflow in design.

Case Study Summary

Order component became more prominent, thought orders for bars only but for other places. Said he tossed around different ideas but went with table structure. Said reasons why table structure. Goes to a document they showed a customer. Has 2 alternatives both technically feasible, one better for business reasons. Going through ideas about how to flag certain columns. Goes to an email to read up something Jason told them. Calls business manager to follow up what he remembers from customer. Asks if something makes sense to business manager. They go through a scenario, rationalizing. He goes to another email. Says "Fine we'll just do it by size code then"

Observations – Company B, CS2.2B

Design Problem; BUG FIX: Multiple bills per day for same customer.

Case Study Summary

Developer1, Developer2 and Customer Representative had meeting with customer and told technical manager about problem. Developer1 makes a suggestion but points out a problem. Someone suggests roll log, Developer1 says easiest is to filter by time. Developer2 argues that point cause its not discrete. Customer won't let them do invoice per day. Developer1 aggress with Developer2. Technical manager makes suggestion about time field but notices a problem. Leave that for now, what's next problem. Customer Representative and Developer1 explain. They mention one solution and discuss it. Technical manager doesn't understand asks question, Developer1 answers. Developer2 brings up time pressure, Technical manager says last point is ad-hocable. They debate if they have time for it. Customer Representative and Developer1 explain 3rd problem. Bill of lating problem, they debate product/domain knowledge. Technical manager asks what customer is expecting – customizable or pre-built. Customer knows its customizable. Ask what problems a 3rd party is giving. Technical manager says do time option and if it overlaps then too bad. They mention one consequence: 3rd party has to do something. Developer1 asks what repercussions are. Double counting and overcharging. Technical manager corrects not overcharged, just repeated, end of day same amount and says "most

likely resistance" is people want to see manifest. A business owner overhearing the conversation says not acceptable to have numbers messed up. What about having customer only do it twice a day? Technical manager says won't resolve the customer being upset Developer1 summarizes, Technical manager suggests a rule, Developer1 argues against. They talk about another idea. Technical manager asks if they want to do it within the timeline. Customer Representative says wait till hear from 3rd party guy.

Observations – Company B, CS2.3B

Design Problem: FEATURE REQUEST: Try to create filtering on a form, how to not be redundant between table and wizard

Case Study Summary

Makes some components in Viso, for the UI. Looks in another application to see how filtering is done, to be consistent. Looks at 2 old ways they did filtering, says they could do it better. Describes his new way. Says they could use the same thing at the top. Enters some values in drop down box. Gets sheets the customer uses for work to see what fields are. Puts notes next to columns of table, says he's unsure about relevance of a column. Changes some of table. Says he's struggling with how to put values in columns given customer type Worried values will be so frequently incorrect customer won't use it. Looks frustrated, goes to Design Discussion minutes on the wiki. Says funny having print available under something other than File, but worry about that later. Makes list saying customer currently prints by cases, kegs & floor. Says current design doesn't seem like the solution to him Says he could design according to what they currently have but knows there's a better way that shows understanding of business. Has to make sure his way doesn't mean they can't do something they currently can. He debates whether or not to put it in a wizard. Because wants customer to know where data is, current setup not great. Talks about model of system cases and kegs really different so has to be a wizard. Says wizards are a lot of clicking and long time to code. Makes 2nd screen of wizard. Says worry about grammar later. Asks developers a technical question and looks on Laszlo site. Makes a 3rrd page of wizard, looks at customer documentation, reduces to 2 pages. Realizes customer can filter in wizard and in table which isn't good. Says would be good to have auto filter like excel. Wonders if a scenario would ever arise where this would be a problem. Asks business manager, who said possibly. One other developer says could have a filter button. This developer says so many options, to an open room of developers and no one really responds. Goes online to Laszlo's documentation. Tries a drop down box and says "something like that". Wants to merge table with load plan but says no space. Says he tries to follow minimalist principles and it seems silly to have things twice. Wants to turn truck around, but wants it to appear the way it does when printed. Could do mouse-overs but doesn't know how much customer needs it at a glance. Thinks, says "you could have some interesting behaviour, you could…" and describes. Says "That's not bad. Not bad actually." Says minimize/maximize takes advantage of Laszlo functionality. Goes to Laszlo documentation to figure out how to do it.

Observations – Company B, CS2.4B

Design Problem: BUG FIX in time zones

Case Study Summary
Scans code and recognizes that it handles only 1 order they need it to handle more. Changes code and runs a test that fails. Talks out loud about time zones. Makes note in logbook. Scans code says he didn't know they created an Allocation Record. Changes code says what he thinks will happen, runs test to confirm, a few times. Says it's a problem with tests not code. Looks at another code file and sees same problem. Says problem is starting to manifest itself in a few different areas so he'd like to fix it. Says adding code to implementation is something they don't want to do. Changes code, says why he's changing it, runs test It works, checks code in.

Observations – Company B, CS2.5B
Design Problem: FEATURE REQUEST: Needs to do more detail in interface from email from Jason for Clearbill
Case Study Summary
Has email with feature request, 2 bullet points, doesn't understand. Searches code and looks back and forth to email. Sends email to ask if ORDER_ID replaces ORDER_NO. Looks at code and laughs saying he doesn't even know what 3 variables mean. Says he's going to change code on basis that he understands and says what he thinks it is. Runs system to try to understand. Says "The easiest thing to do is change the tests". Runs test, which fail, and he reads exceptions. Gets an email response from business manager talking about order numbers, says its different than what he thought. Searches emails from business manager and sends follow-up question to him. Changes code back to original says he thinks they're having a bit of a communication issue. Customer Representative comes in and explains order number stuff to Ian, Ian says thanks. Developer outlines idea, to handle order # stuff, in logbook. Developer describes his solution to another developer who is now pair programming with him. Runs test and lets errors go, because they're irrelevant right now and expected. Changes code, runs test, it passes. Developer says business manager has contradicted himself. Business manager comes in, the two developers talk to him about unique IDs. They change code, test fail. Business manager comes over and explains problem, using an example form. Developer talks bout problem. Business manager and developer talk about problem together, developer says redundant to have 3 variables. Business manager explains why and leaves. The two developers talk about adding 3 variables and say each has a different purpose. They change code and run tests. Developer says they need at least 2 variables, the third is redundant. Says "But we'll leave it in there for now cause it isn't a big deal".

Observations – Company B, CS2.6B
Design Problem: FEATURE REQUEST: How to list things in a table.
Case Study Summary
They look at code form another project with another customer to see how they did a similar problem. Developer1 suggests to just take existing table and change it. Read code, talk about problem "how else do you think it should be?" Read Google stuff and talk about

problem saying they could go by month. Before they just left the problem but "it's a bit silly". Customer Representative provides his idea to do 01Jan, Developer1 doesn't like it. They look on SQL pages and talk about customer documentation. Developer2 raises a problem he has with each line is part of an order. Customer Representative understands problem but wants them to try to stay consistent with what they showed customer. Develoepr1 and Developer2 check with Ian about what happens if they change his code. Developer2 says "Each row will be customer by month, no… each row will be customer by day". Developer2 explains problem of double counting to Developer1. Developer2 provides an idea. Developer2, Developer1 and Customer Representative talk about the problem. Developer1 brings up problem with month by month. Developer2 says have some problem for day by day. Customer Representative says do whatever takes up least time, and leaves. Developer1 proposes a solution but Jay doesn't understand. Developer2 proposes a solution but Developer1 doesn't understand so Developer2 explains. Develoepr1 agrees, Developer2 mentions downside. They go with Developer2's solution to implement.

Observations – Company B, CS2.7B

Design Problem: FEATURE REQUEST: How to order months using ordinal

Case Study Summary

Developer1 brings up an idea he remembers and Developer2 says he remembers it too. They go to Google and a Microsoft site, Mandarin site, MSDN site, back to Mandarin. Look at an example of order and ordinal, highlight code. Say "here's what I wanted". Mandarin is the framework for doing the cub design that they are using in their software. They put that code into their system, change as required. Test it and it works.

Observations – Company B, CS2.8B

Design Problem: FEATURE REQUEST: Set rule into charge, create another cube, set last modified task

Case Study Summary

They talk about doing it one way, Developer1 says "or you could…" and explains. Developer2 agrees and they change a method, run the test and it fails. Developer2 says why he thinks the test failed and Developer1 agrees. Developer2 raises a problem and Developer1 raises a solution then stops himself saying why they can't. They suddenly realize you can have 2 rules with the same name. They tell Customer Representative who asks if it's a big problem and says he doesn't thinks its an issue. Developer2 suggests to leave it because they can't really fix it. Developer1 says "crap. This was so perfect too." They debate whether or not to leave it, run a scenario of system. Developer2 brings up an idea, Developer1 agrees. They change code, insert comments, say "I think that's the best we can do". Run tests, which fail. Developer2 makes a suggestion about what to do, so does Developer1. Developer2 changes code, they run tests, it works.

Observations – Company B, CS2.9B

Design Problem: FEATURE REQUEST: How to load beer on a truck.

Case Study Summary

Technical manager said what he thought the customer did, Developer1 said greedy algorithm was the fastest. Technical manager said it was like a different project they had worked on. Developer1 talked to a customer who said you have to be there to see it. 2 Problems: how to stack for ordering route, how to stack for weight. Developer1 tried to put together if conditions, but too many combinations. Developer1 is doing a literature review of all different algorithms. Originally thought only 1st problem important, now #2 more prominent. … Customer Representative met with customer to discuss some things. Customer said to go with one ID # but not that simple. Customer Representative uses order form like what Customer would have. Talked about inconsistencies between order from and what Customer said. Customer Representative tells Developer1 to worry about bottles only, not cans, right now. Customer doesn't know how they want to organize cans yet. They go over ways to optimize by location or by weight. … Developer1 researches, an algorithm
He is going to meet with customer to learn more and "show face". They tried a few different algorithms. Developer1 also took an AI class and is interested in the topic. They tried linear programming, they tried going to a university professor, they tried if/else approach, they tried some greedy algorithm approaches and genetic algorithm approaches and linear programming.

Observations – Company B, CS2.10B
Design Problem: BUG FIX: Capitalization looks careless
Case Study Summary
Developer1 starts changing capitalization, Developer2 says "whoa man' and tells Developer3 that Developer1 is changing code. Developer3 says "I wouldn't do that because…" Developer1 says they could lower case them all. Developer3 says he'd avoid that. Developer1 says he thought it'd be an easy bug fix that's why he picked it. Developer1 asks why he can't change case, Developer2 says cause XML doesn't ignore case. Developer1 says if all test pass he still can't change it? Developer2 says he'd feel confident if they had an end to end test. Developer2 says if Integrator doesn't use it they're probably ok.. They debate whether or not to change it. Developer1 says its too low priority and doesn't make change. Writes comment in Bug Report interface.

Observations – Company B, CS2.11B
Design Problem: FEATURE REQUEST: Adhoc rules fire charges but aren't based on a built in rule so they don't work with the Enterprise Transaction Layer
Case Study Summary
Developer1 says they have to create a rule dimension and James asks if they have to. Developer1 says lets think of another way. They write out 2 scenarios on a piece of paper and talk about problem. Say "I don't know" a lot and rationalize it with each other. They debate which way to go of the two approaches. Developer2 says one approach they don't have to change anything; They talk about a scenario they don't' know about. Customer Representative comes in and they bring up the problem with him. Their group conversation goes through alternatives and goods and bads of each. Customer Representative asks what the easiest thing to do is, and its to not charge, but they talk about downside. Developer2 brings up a problem scenario. They talk about scenarios and Developer2 asks what an ideal

situation is. Developer2 says what the ideal would be. Customer Representative asks about time pressure. They bring up more scenarios and solutions and goods and bads. Customer Representative says he's going to push for the customer to get their business down. They talk about more scenarios and solutions. Customer Representative asks about time pressure. Developer1 draws out a solution and explains it and they say that's not bad. They talk about specific scenarios with the idea Developer2 had, confirming that it works. Customer Representative says it satisfies the customer, just tight for time. They start to work on solution then realize it will take them longer than they though. They'd have a lot of time invested for something that's used 5% of the time. They explain Developer2's idea to Customer Representative and good and bad and time. It fits time pressure so that's what they do.

Observations – Company C, CS2.1C

Design Problem: FEATURE REQUEST: Figure out what to do because client doesn't have an SDK and company's integration pack needs an SDK to talk to.

Case Study Summary

Tires to contact someone to ask about it. Says supposed to be a solution over SQL but that seemed rudimentary. Says never run into a situation where there was no SDK. Says if he can find out about the SDK then they don't' need a consultant.

Observations – Company C, CS2.2C

Design Problem: FEATURE REQUEST: Make a message pop up to a user that an event happened in one of their products

Case Study Summary

Reads documentation open for a subsystem. Changes directories in DOS, copies items in explorer. Says a 3rd party monitoring software he is looking at is similar in purpose to theirs, different in architecture. They each have a monitor (a.k.a probe) as an executable.

He said seems like their architecture wouldn't scale very well. Says "Their choice." Reads an error log and doesn't understand "Whatever that means."

…

Says 2 sets of instructions, one doesn't apply the other doesn't' makes sense. Tries to figure out what a probe binary directory is, from the documentation. Reads a section called "Using a specific probe." which is a PDF. "After every possible number of attempts I now go to the knowledge base." And opens Google. Says the problem is the system can't find the file it needs to find. Submits request to technical support for sub-system. "And then something just hit me" he says about an idea, when trying another probe. Explains they have a different path format for windows paths. Tries it, it doesn't work. Sets up a sandbox to do the email probe. "The only thing that concerns me is this filter field. I don't know how it works. Gets an email from technical support asking him to run system in debug mode. Goes back and forth asking how to run in debug mode. "I guess its running . I have no idea."

Observations – Company C, CS2.3C
Design Problem: BUG FIX
Case Study Summary
Reads source code in a text editor. Looks frustrated when reading the code. Asks another developer a question. The other developer runs his system and this developer tells him to run it without the debugger. This developer directs him to a "variance" variable but the other developer doesn't have a "variance" variable in his version. This developer asks him to try something else and they recognize a problem in the system. They talk about the problem and this developer says he can fix it. He shows the other developer his code and they talk about the design. This developer says they need to extract a method, the other developer says now he understands why they needed recursion.

Observations – Company C, CS2.4C
Design Problem: Whether or not can use .NET
Case Study Summary
Developer1 tells Technical Lead they need to decide if they want to use .NET. Technical Lead says that comes later, or from a different angle. When they talk about environment for web services Developer2 says .NET or GSoap. Developer2 says .NET cause it has cleaner interfaces, Technical Lead asks what the limitations are. They talk about why to use .NET, its more sophisticated. Asks what else they have that uses .NET, one application. Technical Lead says reason didn't have .NET was cause management wanted to avoid it. Technical Lead says not their decision and can't make decisions that are contradictory. Technical Lead says that decision will come tomorrow.
…
Developer1 presents 2 implementation options for his application to Customer Representative and Technical Manager: C++ & GSoap or .NET. One Customer Representative says they're not fussy if they use .NET. Developer2 talks about version issues. They talk about advantages and disadvantages with .NET and GSoap. Customer Representative says he'll check with Kay, doesn't see it being a problem. Suddenly Technical Manager says its HIS call, that the Technical Lead should have talked to him. Technical Manager announces that they can use .NET.

Observations – Company C, CS2.5C
Design Problem: BUG FIX: Changing a UI to make inputs more constrained by putting a drop down box instead of a field. "Other" vs. "other"
Case Study Summary
Company wants drop down boxes instead of fields. Developer said QA said system was awkward. Reads documentation, runs system to see how it works. Realizes what function is the problem. Says has to learn how Forms in a subsystem work. Runs another system to understand. "One difference between what we do and what this other subsystem does…" Goes to notify method and says he's going to change it. … Sorts through source code in subsystem domain. Explains next step to me about how he makes the change.

APPENDIX B: GLOSSARY

Action Research	"Inquiry within organizations aimed at learning, improvement and development." [Patton, p 179]. "A spiral of interlocking cycles of planning, acting, observing and reflecting." [Schwandt, p3].
Boundedly Optimal	Defining a concept as good or best, recognizing the subjectivity of good or best; Defining a concept as right or good within a subset of potential alternatives, that is relative to the environment in which the evaluation occurs. This second definition blurs the boundary between optimal and boundedly optimal.
Case	"a specific and bounded (in time and place) instance of a phenomenon selected for study, characterized by their correctness and circumstantial specificity and their theoretical interest of generalizability" [Schwandt, 22].
Content Analysis	A research methodology that examines words or phrases within a wide range of text [Colorado ref]
Design (of software)	The organization and arrangement of domain-specific real world concepts and artifacts in a software application at varying and potentially intersecting levels of abstraction.
Design Change	A modification to the existing design of a software application requiring one or more implicit or explicit decisions.
Design Decision	The selection of an option among zero or more known and unknown options concerning the design of a software application
Emergent	"Openness to adapting inquiry as understanding deepens and/or situations change." [Patton, p40]. Refers to the experimental design.
Empiricism	"This is the name for a family of epistemological theories that generally accept the premise that knowledge begins with sense experience." [Schwandt, p67].
Explicit Knowledge	Knowledge that is easily communicated or codified [Polyani]
Hypothesis	"Sentences or statements that express what we believe, doubt, affirm or deny" [Schwandt, p211]. In quantitative experimentation hypotheses are defined at the beginning of the controlled experiment and contribute to the structure of the overall research design. In qualitative experimentation hypotheses are allowed to emerge during the experimentation, potentially as a result of the experiment.
Knowledge	A cognitive representation of a concept held by a person (the "knower") and shareable with other knowers.
Market Knowledge	Tacit or explicit knowledge where the quantity and representation are controlled by someone other than the knower, and which impacts the knower's structural knowledge when the knowledge domains intersect. E.g. demographic pressure that sales of a book increase when happy endings are provided.
Mental Model	A model that "represents entities and persons, events and processes, and

	the operations of complex systems" by satisfying the principle of iconicity, the principle of possibilities, and the principle of truth. The principle of iconicity states that, "a mental model has a structure that corresponds to the known structure of what it represents". The principle of possibilities states that, "each mental model represents a possibility". The principle of truth states that, "each mental model represents a true possibility and it represents a clause in the premises only when the clause is true in the possibility." [Johnson-Laird]
Model	A representation of a concept of system upon which the maintainer of the model can run simulations and pos and answer questions about the concept or system being modeled [Bruegge, p12]
Naturalistic Inquiry	"Studying real world situations as they unfold naturally, non-manipulative and non-controlling." [Patton, p40]. Refers to the experimental design.
On-looker Observations	"Direct firsthand eyewitness accounts of everyday social action" [Schwandt, p179] where the researcher is not a predefined member of the social action observed or an active participant in the social action that does occur. "The extent of participation is a continuum that varies from complete immersion in the setting … to complete separation from the setting" [Patton, p265].
Optimal	At an extreme, optimal is defining a concept as right, according to an objective definition of right or correct, or according to a widely-accepted subjective definition of right or correct. This second provision (i.e. widely accepted subjective definition) blurs the boundary between optimal and boundedly optimal. At a non-extreme, optimal is definitng a concept as the *most* favourable or *most* desirable, under a given set of conditions. It is this non-extreme use of the term that also overlaps with the concept of Boundedly Optimal.
Participatory Observations	"The notion of 'being there', of witnessing social action firsthand, emerged as a professional scientific norm" in the environment studied [Schwandt, p185]. "It requires that the researcher … take some part in the daily activities of the people among whom he or she is studying." [Schwandt, p186].
Premise	A statement about a problem that conveys semantic information which shapes the problem [Johnson-Laird, Simon]. This term is used in this thesis as a statement to shape a mental model. This term could be used interchangeably with the term "proposition" but in this thesis this is not done on a regular basis.
Proposition	"Sentences or statements that express what we believe, doubt, affirm or deny. Assertions, interpretations, hypotheses … findings and the like are familiar kinds of propositions or statements" [Schwandt, p211]. They are used to contribute to the building of an argument or explanation for an examined phenomenon.
Qualitative	"Many scholars use the phrase qualitative inquiry as a blanket designation for all forms of social inquiry that rely primarily on qualitative data (i.e. data in the form of words), including ethnography,

	case study research, naturalistic inquiry. … To call a research activity qualitative inquiry may broadly mean that it aims at understanding the meaning of human action." [Schwandt, p213]. In this thesis the term qualitative is used to refer to both qualitative inquiry and qualitative data, depending on the context of the sentence. Where used to refer to qualitative data, there may also be 'subjective'.
Quantitative	"This is an adjective indicating that something is expressible in terms of a quantity (i.e. a definite amount or number). … The term is often used as a synonym for any design (e.g. experimental and survey) … that relies principally on the use of quantitative data." [Schwandt, p215]. In this thesis the term quantitative is used to refer to both quantitative inquiry and quantitative date, depending on the context of the sentence. Where used to refer to quantitative data, there may also be "objective".
Representation	Resemblance, replication, repetition, description, duplication [Schwandt, p227]. In qualitative inquiry representation is attempting to depict social phenomena, and the crisis of representation addresses the ability of a researcher to ever properly and completely understand a social phenomena to the point that it is properly and completely depicted as it is [Schwandt, p 227, 41]
Risk	"The probability that the return on an investment will be more or less than expected. The variability of the return on an investment." [Lasher, p721]. The definitions and values of probability , return, investment and variability change from quantitative to qualitative metrics depending on the domain discussed. E.g In problems of finance these parameters and values are based on cost and statistical probability. In problems of health these parameters are based on genetics, chance and other such indeterminants.
Small Software Team or Organization	A group of less than 20 people developing software
Structural Knowledge	Tacit or explicit knowledge where the quantity and representation are controlled in large part by the knower. E.g. an author of a story shapes the contents of the story.
Tacit Knowledge	Knowledge that is often difficult to make explicit because it grows out of habit, culture, experience and other context-specific factors [Polyani66]
Uncertainty	A degree of knowledge that is deemed insufficient in some component of the represented concept, potentially introducing risk onto the situation that uses that knowledge.
Understand	"To comprehend the meaning of some [knowledge]… and to risk that we might misunderstand – that is, that we might fail to comprehend the meaning of some [knowledge]" [Schwandt, p262].
Unit of Analysis	Parameters defined by the research design that characterize the type of the object(s) examined, and which constitute a portion of the sample of the experiment. In the blurring of qualitative and quantitative

	empiricism, the term unit of analysis is used interchangeable with the term ‘case” [Patton p226, Yin p24]
Window Size	The amount of text included in a single code during content analysis. In general, a smaller window leads to less context than a larger window.

INDUCTIVELY GENERATED CODE DEFINITIONS

Code	Definitions
Adaptation	Modification to an entity or situation. E.g. change, fix, accommodate
Age	Representation of youthfulness or seniority of an entity or person. E.g. old, new, first
Alternatives	More than one option for a situation. E.g. either, alternatives, debating among.
Better vs. Worse	A quality existing on a continuum bounded by Good and Bad.. E.g. dysfunctional, *good* book, *bad* design.
Code	Reference to implementation, code, or writing software. E.g. written, coding, program.
Communications	The participation in or occurrence of the exchange of information. E.g. Talked, asked, discussed
Customer Product	Events, entities and concepts tied to the persons for whom the software is created. E.g. project, domain, feature.
Decision	Selection of an alternative among zero or more alternatives. E.g. design decision, decision making, indecisive.
Desire	Expressing wanting to do something or preference to a certain entity or approach. E.g. want, prefer, desire
Different	Concept or entity being different from another. E.g different, variations, odd
Difficult	Challenges in accomplishing tasks or understanding concepts. E.g. easy, hard, complex.
Documents	Information recorded on any medium. E.g. document, wrote down, recorded
Educate	Learning or teaching a skill, information, concept or process. E.g. read, learn, master.
Experience	Prior events or happenings brought to the current situation. E.g. expert, done that before, experience
Experiment	Attempting a situation without full knowledge of the outcome. E.g. gave it a shot, tried that out, experimented
Externals	Outside people/issues that may or may not impact a situation. E.g. marketing, pressure, management
Feedback	Information in response to a prior event or situation. E.g feedback, response, that told us
Frequency	Number of occurrences or amount of time. E.g. many times, not much, always.
Group	Over 1 person. E.g. we, team, group.

Idea	Thought, opinion or concept held by an individual or group. E.g. think, concept, ideas.
Information	Applied knowledge of a situation. E.g. information processing, criteria, accumulate information
Knowledge	Knowledge of a concept, often referable to a source. E.g. know, book, Gang of Four
Model	Representing some concept. E.g. convey, bad *design*, patterns.
Need	A "must have". E.g. need, had to do that, requirement.
Personal Attributes	Positive and negative descriptive qualities of a person or entity. E.g. verbose, *spaghetti* code, cool.
Pre-Existing	Existing prior to current state. E.g. already had, the way it was before, with what was there
Problem	An event or situation requiring resolution. E.g. problem, issue, situation.
Recognition	A concept or situation seemingly clear to a perceiver. E.g. explicit, the same, seem.
Re-Do	Execution of some activity more than once E.g. reviewed, re-think, re-did
Right vs. Wrong	Identification of something as being part of a right or wrong dichotomy. E.g. solution, right, wrong.
Satisfice	Identification of something as satisfactory. E.g. enough, satisfactory, sufficient.
Search	To research, information or knowledge. E.g. Buy books, mechanical discovery, find.
Size	How big or small an event, situation, entity or person is. E.g. incremental, total, lightweight
Skill	Ability E.g. skill, capabilities, ability
System Breakdown	Breakdown of some modeling concept or task. E.g. isolation, separation, part of the
System Integration	Integration of some modeling concept or task. E.g converge, attach, integrate
Technology	Tools and materials used to achieve some task. E.g. "XSLT", "XML", "Java".
Time	Amount of time allocated to an event or situation. E.g. now, at the time, this week.
Understand	Perceived meaning of a concept or situation. E.g. understand, meaning, makes sense
Utility	The usefulness of an entity, situation or person. E.g. take advantage, useful, utility

Work Path Length	Identifying time surrounding some situation which has a start, middle and end. E.g. beginning, done, started
Work Processes	Events, entities and concepts tied to the process by which software is developed. E.g. version, refactoring, agile.