Preface

The most distinctive thing about humans is not the thumb, of course. It is design. Unlike any other animal, we incessantly and dramatically reshape both ourselves and our environment. We design ourselves through innovating concepts, language, culture, and other practices, and we design almost everything around us. It is telling that we now speak of "natural" places on the Earth to distinguish the few places we have not (yet!) redesigned.

Among the most complex, diverse, and pervasive things that humans design are software systems. The history of software design is almost entirely a history of trying to catch up with complexity and diversity. As we look back to the 1960s the notion of what was then called the "software crisis" seems almost amusing. At that time, barely a decade after the invention of software, it was recognized that the complexity and diversity of software systems was being elaborated far more rapidly than were engineering techniques to manage software development. What is amusing is that this was (optimistically) called a *crisis*, as if it were a temporary threat that would in the course of time be rectified.

But this never happened. Instead the software crisis became chronic. It became the context for the software industry and for software engineering. And by now, as almost every system is, incorporates, or fundamentally depends upon software, as software systems have become utterly pervasive, the software crisis has really become an epoch in human history.

No one is very happy about this, and from time to time manifestations of the ongoing software crisis bubble up into dramatic mass media reports about how vital defense systems are fundamentally unverifiable, about how medical systems make it more or less inevitable than surgeons will kill their patients, about how banking systems occasionally share account information with unknown hackers, and so forth.

What are we to do? There are many answers, many approaches, but none of them is a "silver bullet" (as Fred Brooks vividly put it). The most obvious approach, and quite likely the most powerful, is to explicitly describe and justify the design, implementation, and use of software systems, and to do this routinely, iteratively, and regularly throughout the software development process. We call this "Rationale-Based Software

Engineering." It is not a new idea, though in some areas there are new tools and techniques. Rather, it is an essential idea that has been around, that we cannot afford to lose track of, and that perhaps can be pushed to greater fruition now. In this book, we try to bring together a broad discussion of rationale and focus on aspects of the very old and very weighty challenge of the software crisis.

Book Overview

This book consists of four parts. Part 1 sets the context for the work and describes why Software Engineering Rationale (SER) and Rationale-Based Software Engineering (RBSE) are essential contributors toward improving the software development process. Part 2 describes how Software Engineering Rationale can be used to support software development. Part 3 describes how RBSE can be applied throughout the software engineering lifecycle as well as supporting software reuse. Part 4 presents architectural and conceptual frameworks for RBSE as well as our vision of future directions for RBSE research.

Part 1: Introduction

So why capture rationale? Before making a case for why SER capture and use should be an essential part of software development, it is important to first define what it is. Part 1 defines rationale and sets the context for the remainder of the book.

Chapter 1, "What is Rationale and Why Does it Matter" provides an initial discussion of the scope and value of rationale in software engineering. An initial introduction of previous work on rationale is provided and we make our initial case for why rationale is useful during software engineering.

Chapter 2, "What Makes Software Different" describes some of the key differences between applying rationale to software engineering and applying rationale to other domains. This includes both opportunities for use in software engineering that are lacking when developing other artifacts as well as some of the unique challenges posed by software development. Specifically, we look at the role of the computer in software development versus physical artifact development as well as the implications of the necessity to support iteration in software development on rationale management.

Chapter 3, "Rationale and Software Engineering" introduces both Software Engineering and Software Engineering Rationale (Dutoit et al.

2006b). Rationale has a role to play in defining software processes, supporting software project management, and as a mechanism to both document and guide decision-making throughout the software process.

Chapter 4, "Learning from Rationale Research in Other Domains" describes key rationale research in other domains and its implication to software engineering. The chapter focuses on four areas: domain-oriented design environments using Procedural Hierarchy of Issues (PHI) (McCall 1991); automating design rationale capture in Computer-Aided Design, more specifically that using the Rationale Construction Framework (Myers et al. 1999); rationale support via Parameter Dependency Networks and DRIVE (de la Garza and Alcantara 1997); and how Case-Based Reasoning (CBR) systems such ARCHIE (Zimring et al. 1995) relate to rationale.

Chapter 5, "Decision-Making in Software Engineering" examines the role that human decision-making has in software engineering. The chapter describes naturalistic decision-making and Klein's recognition-primed decision model (Klein 1998), which addresses some of the problems with classical decision making by proposing a strategy more consistent with observations of human decision-makers, where the first acceptable alternative is selected. The chapter concludes with a discussion of rationale as a resource for decision-making and how rationale relates to both the classical and naturalistic views.

Part 2: Uses of Rationale

There is little or no point in capturing rationale if there are not ways in which it can be used. Part 2 describes some key uses of rationale in software development.

Chapter 6, "Presentation of Rationale" looks at rationale presentation. The two major classes of presentation formats, semiformal and informal, are described. The chapter then describes new opportunities for presentation provided by reusable rationale databases, multiscale presentation, and development tool integration.

Chapter 7, "Evaluation" describes how rationale can be used for evaluation from two angles. The first is how argumentation-based rationale can be used for decision evaluation by evaluating the consistency and completeness of the rationale as well as evaluating support for development alternatives taking into account decision criteria, input from multiple developers, and uncertainty. The second approach to evaluation describes scenario-based evaluation as supported by scenario-based design (Carroll and Rosson 1992).

Chapter 8, "Support for Collaboration" discusses rationale and collaboration from two perspectives. The first is how the highly collaborative nature of software development supports the development, codification, and use of rationale. The need for collaborators to justify their decisions to each other is a key source of rationale. The other is how rationale supports collaboration by encouraging the exchange of information and awareness of the goals of team members.

Chapter 9, "Change Analysis" identifies the important role that rationale can play in assessing the impact of changing requirements, design criteria, and assumptions on a software system. By explicitly recording the impact that those elements had on the decisions involved and relating the results of the decision-making process to the artifacts that instantiate them, the rationale can be used to detect where changes will be required if requirements, criteria, and assumptions change. In addition, rationale can also capture crucial inter-decision dependencies and alert the developer if one of those dependencies is later violated.

Part 3: Rationale and Software Engineering

In software engineering, decision-making is not restricted to only part of the process. There are critical decisions to be deliberated throughout the lifecycle of the software system. Part 3 describes how rationale supports the various stages of the software lifecycle and how rationale research relates to other software engineering research that also supports those stages.

Chapter 10, "Rationale and the Software Lifecycle" gives a brief introduction to the stages of software development and how rationale can be utilized. The topic of lifecycle modeling is then introduced and the application of rationale to sequential models, such as waterfall and the v-model is described as well as how rationale can be applied to iterative approaches. The chapter concludes with a discussion of how rationale supports process improvement initiatives.

Chapter 11, "Rationale and Requirements Engineering" describes rationale's contribution to requirements engineering. This includes how rationale can support the requirements definition process by assisting with requirements elicitation, achieving consensus on requirements, identifying requirements inconsistency, and supporting requirement prioritization. Rationale's role in requirements traceability and the relationship between rationale and nonfunctional requirements is also described. The chapter concludes with how rationale can assist in adapting to changing requirements, one of the major challenges in software engineering.

Chapter 12, "Rationale and Software Design" describes design rationale as applied to software design. The chapter begins with a description of the nature and importance of software design rationale, both that generated by the designers while designing and that generated during construction and use. Two fundamentally different types of decisions are described—design space decisions and rationale for non-design-space decisions that represent a deeper reflection on the design process. We conclude with a look at some specific approaches to rationale as applied to software design and software architecture.

Chapter 13, "Rationale and Software VV&T" defines verification and validation and then describes the issues involved in the major types of software tests—inspection, unit testing, integration testing, and system testing. The role of rationale in software testing is described by focusing on three major uses: the contribution of rationale to testability, rationale's contribution to test case prioritization, and using rationale to support component testing and selection.

Chapter 14, "Rationale and Software Maintenance" describes how rationale can be used to support software maintenance. The chapter describes four areas where rationale can support maintenance: maintenance prediction, impact assessment, program comprehension, and maintenance rationale. The chapter then concludes with a discussion of why rationale should also be captured during software maintenance and some existing research that supports the capture of maintenance rationale.

Chapter 15, "Rationale and Software Reuse" begins with a description of key software reuse concepts and categories, along with defining types of rationale that support reuse. The chapter then describes several ways that rationale has been, or can be, applied to assist with software re-use.

Part 4: Frameworks for Using Rationale in Software Engineering

In this part, we take a look ahead. In order to support Rationale-Based Software Engineering, it is necessary to have frameworks to define the key concepts and architectural needs for Rationale Management Systems. In this part, we define a conceptual framework and architectural framework to support Rationale-Based Software Engineering.

Chapter 16, "A Conceptual Framework for Rationale-Based Software Engineering" describes the goals of conceptual frameworks in general, followed by what is needed by a conceptual framework for rationale use in software engineering. To support the decision-centric approaches, we define a taxonomy of software decisions that could be answered using

SER. To support usage-centric approaches, we describe how Carroll and Rosson's (1992) Scenario Claims Analysis (SCA) rationale can be applied to software engineering. We conclude with a discussion of the implications of iteration, a summary of current challenges to rationale use, and propose some potential solutions.

Chapter 17, "An Architectural Framework for Rationale-Based Software Engineering" describes the key features needed for a Rationale Management System (RMS) to support software engineering. This includes the model management subsystem (which includes support for capture and formalization), the underlying hypermedia substrate, and the necessary integrations between RMS and external software development support systems.

Chapter 18, "Rationale-Based Software Engineering: Summary and Prospect" serves two purposes. First, it summarizes the work presented in this book and its implications for future rationale research and use. We then look at some key future challenges to software development and conclude with a discussion of both the promises of and challenges to Rationale-Based Software Engineering.

Acknowledgements

This book would not have been possible without the support of many people. First of all we would like to thank Ralf Gerstner of Springer, Germany for making this project possible and for invaluable advice in publishing matters. We appreciate his infinite patience with watching this project come to fruition. We would also like to thank Bashar Nuseibeh and Colin Potts for their excellent and inspiring forewords. At Miami University, Monica Baxter provided secretarial assistance in pulling together the various components of the book. Last, but not least, we would like to thank our friends, family, and colleagues for their support, patience, and encouragement throughout this project.

http://www.springer.com/978-3-540-77582-9

Rationale-Based Software Engineering
Burge, J.E.; Carroll, J.M.; McCall, R.; Mistrík, I.
2008, XXXV, 316 p., Hardcover
ISBN: 978-3-540-77582-9