Praise for *Just Enough Software Architecture: A Risk-Driven Approach*

If you're going to read only one book on software architecture, start with this one. *Just Enough Software Architecture* covers the essential concepts of software architecture that everyone — programmers, developers, testers, architects, and managers — needs to know; and it provides pragmatic advice that can be put into practice within hours of reading.

—Michael Keeling, professional software engineer

This book reflects the author's rare mix of deep knowledge of software architecture concepts and extensive industry experience as a developer. If you're an architect, you will want the developers in your organization to read this book. If you're a developer, do read it. The book is about architecture in real (not ideal) software projects. It describes a context that you'll recognize and then it shows you how to improve your design practice in that context.

—Paulo Merson, practicing software architect and
Visiting Scientist at the Software Engineering Institute

Fairbanks' focus on "just enough" architecture should appeal to any developers trying to work out how to make the architecting process tractable. This focus is made accessible through detailed examples and advice that illustrate how an understanding of risk can be used to manage architecture development and scope. At the same time, Fairbanks provides detail on the more academic aspects of software architecture, which should help developers who are interested in understanding the broader theory and practice to apply these concepts to their projects.

—Dr. Bradley Schmerl, Senior Systems Scientist, School of
Computer Science, Carnegie Mellon University

The Risk-Driven Model approach described in George Fairbanks' *Just Enough Software Architecture* has been applied to the eXtensible Information Modeler (XIM) project here at the NASA Johnson Space Center (JSC) with much success. It is a must for all members of the project, from project management to individual developers. In fact, it is a must for every developer's tool belt. The Code Model section and the anti-patterns alone are worth the cost of the book!.

—Christopher Dean, Chief Architect, XIM,
Engineering Science Contract Group – NASA Johnson Space Center

Just Enough Software Architecture will coach you in the strategic and tactical application of the tools and strategies of software architecture to your software projects. Whether you are a developer or an architect, this book is a solid foundation and reference for your architectural endeavors.

—Nicholas Sherman, Program Manager, Microsoft

Fairbanks synthesizes the latest thinking on process, lifecycle, architecture, modeling, and quality of service into a coherent framework that is of immediate applicability to IT applications. Fairbanks' writing is exceptionally clear and precise while remaining engaging and highly readable. *Just Enough Software Architecture* is an important contribution to the IT application architecture literature and may well become a standard reference work for enterprise application architects.

—Dr. Ian Maung, former Senior VP of Enterprise Architecture at Citigroup and Director of Enterprise Architecture at Covance

This book directly tackles some key needs of software practitioners who seek that blend of tools to help them create more effective systems, more effectively. George reaches frequently into his own experience, combining important ideas from academia to provide a conceptual model, selected best practices from industry to broaden coverage, and pragmatic guidance to make software architectures that are ultimately more useful and realistic. His simple risk-based approach frames much of the book and helps calibrate what "just-enough" should be. This book is an important addition to any software architecture bookshelf.

—Desmond D'Souza, Author of MAp and Catalysis, Kinetium, Inc.

This book shows how software architecture helps you build software instead of distracting from the job; the book lets you identify and address only those critical architectural concerns that would otherwise prevent you from writing code.

—Dr. Kevin Bierhoff, professional software engineer

System and software developers asking questions about why and where about software architecture will appreciate the clear arguments and enlightening analogies this book presents; developers struggling with when and how to do architecture will discover just-enough guidance, along with concepts and ideas that clarify, empower, and liberate. All in all, this book is easy to read, concise, yet rich with references — a well-architected and finely-designed book!

—Dr. Shang-Wen Cheng, flight software engineer

Just Enough Software Architecture

A Risk-Driven Approach

George Fairbanks

Marshall & Brainerd Publishing
2445 7th Street
Boulder, CO 80304
(303) 834-7760

Library of Congress PCN

ISBN 978-0-9846181-0-1

First printing, August 2010

Foreword

In the 1990s software architecture emerged as an explicit subfield of software engineering when it became widely recognized that getting the architecture right was a key enabler for creating a software-based system that met its requirements. What followed was a dizzying array of proposals for notations, tools, techniques, and processes to support architectural design and to integrate it into existing software development practices.

And yet, despite the existence of this body of material, principled attention to software architecture has in many cases not found its way into common practice. Part of the reason for this has been something of a polarization of opinions about the role that architecture should play. On one side is a school of thought that advocates architecture-focused design, in which architecture plays a pivotal and essential role throughout the software development process. People in this camp have tended to focus on detailed and complete architectural designs, well-defined architecture milestones, and explicit standards for architecture documentation. On the other side is a school of thought that deemphasizes architecture, arguing that it will emerge naturally as a by-product of good design, or that it is not needed at all since the architecture is obvious for that class of system. People in this camp have tended to focus on minimizing architectural design as an activity separate from implementation, and on reducing or eliminating architectural documentation.

Clearly, neither of these camps has it right for all systems. Indeed, the central question that must be asked is "How much explicit architectural design should one carry out for a given system?"

In this book, George Fairbanks proposes an answer: "Just Enough Architecture." One's first reaction to this might be "Well, duh!" because who would want too much or too little? But of course there is more to it than that, and it is precisely the detailing of principles for figuring out what "just enough" means that is the thrust of this

book. As such, it provides a refreshing and non-dogmatic way to approach software architecture — one with enormous practical value.

Fairbanks argues that the core criterion for determining how much architecture is enough is risk reduction. Where there is little risk in a design, little architecture is needed. But when hard system design issues arise, architecture is a tool with tremendous potential. In this way the book adopts a true engineering perspective on architecture, in the sense that it directly promotes the consideration of the costs and benefits in selecting a technique. Specifically, focusing on risk reduction aligns engineering benefits with costs by ensuring that architectural design is used in situations where it is likely to have the most payoff.

Naturally, there are a lot of secondary questions to answer. Which risks are best addressed with software architecture? How do you apply architectural design principles to resolve a design problem? What do you write down about your architectural commitments so that others know what they are? How can you help ensure that architectural commitments are respected by downstream implementers?

This book answers all of these questions, and many more, making it a uniquely practical and approachable contribution to the field of software architecture. For anyone who must create innovative software systems, for anyone who is faced with tough decisions about design tradeoffs, for anyone who must find an appropriate balance between agility and discipline — in short, for almost any software engineer — this is essential reading.

David Garlan
Professor, School of Computer Science
Director of Professional Software Engineering Programs
Carnegie Mellon University
May 2010

Preface

This is the book I wish I'd had when I started developing software. At the time, there were books on languages and books on object-oriented programming, but few books on design. Knowing the features of the C++ language does not mean you can design a good object-oriented system, nor does knowing the Unified Modeling Language (UML) imply you can design a good system architecture.

This book is different from other books about software architecture. Here is what sets it apart:

It teaches risk-driven architecting. There is no need for meticulous designs when risks are small, nor any excuse for sloppy designs when risks threaten your success. Many high-profile agile software proponents suggest that some up-front design can be helpful, and this book describes a way to do just enough architecture. It avoids the "one size fits all" process tar pit with advice on how to tune your architecture and design efforts based on the risks you face. The rigor of most techniques can be adjusted, from quick-and-dirty to meticulous.

It democratizes architecture. You may have software architects at your organization — indeed, you may be one of them. Every architect I have met wishes that all developers understood architecture. They complain that developers do not understand why constraints exist and how seemingly small changes can affect a system's properties. This book seeks to make architecture relevant to all software developers, not just architects.

It cultivates declarative knowledge. There is a difference between being able to hit a tennis ball and knowing why you are able to hit it, what psychologists refer to as *procedural knowledge* versus *declarative knowledge*. If you are already an expert at designing and building systems then you will have employed many of the techniques

found here, but this book will make you more aware of what you have been doing and provide names for the concepts. That declarative knowledge will improve your ability to mentor novice developers.

It emphasizes the engineering. People who design and build software systems have to do many things, including dealing with schedules, resource commitments, and stakeholder needs. Many books on software architecture already cover software development processes and organizational structures. This book, in contrast, focuses on the technical parts of software development and deals with what developers do to ensure a system works — the *engineering*. It shows you how to build models and analyze architectures so that you can make principled design tradeoffs. It describes the techniques software designers use to reason about medium- to large-sized problems and points out where you can learn specialized techniques in more detail. Consequently, throughout this book, software engineers are referred to as *developers*, not differentiating architects from programmers.

It provides practical advice. This book offers a practical treatment of architecture. Software architecture is a kind of software design, but design decisions influence the architecture and vice versa. What the best developers do is drill down into obstacles in detail, understand them, then pop back up to relate the nature of those obstacles to the architecture as a whole. The approach in this book embraces this drill-down/pop-up behavior by describing models that have various levels of abstraction, from architecture to data structure design.

About me

My career has been a quest to learn how to build software systems. That quest has led me to interleave academics with industrial software development. I have the complete collector's set of computer science degrees: a BS, an MS, and a PhD (the PhD is from Carnegie Mellon University, in software engineering). For my thesis, I worked on software frameworks because they are a problem that many developers face. I developed a new kind of specification, called a design fragment, to describe how to use frameworks, and I built an Eclipse-based tool that can validate their correct usage. I was enormously fortunate to be advised by David Garlan and Bill Scherlis, and to have Jonathan Aldrich and Ralph Johnson on my committee.

I appreciate academic rigor, but my roots are in industry. I have been a software developer on projects including the Nortel DMS-100 central office telephone switch, statistical analysis for a driving simulator, an IT application at Time Warner Telecommunications, plug-ins for the Eclipse IDE, and every last stitch of code for my own web startup company. I tinker with Linux boxes as an amateur system administrator and have a closet lit by blinking lights and warmed by power supplies. I have sup-

ported agile techniques since their early days — in 1996 I successfully encouraged my department to switch from a six-month to a two-week development cycle, and in 1998 I started doing test-first development.

Who is this book for?

The primary audience for this book is practicing software developers. Readers should already know basic software development ideas — things like object-oriented software development, the UML, use cases, and design patterns. Some experience with how real software development proceeds will be exceedingly helpful, because many of this book's basic arguments are predicated on common experiences. If you have seen developers build too much documentation or do too little thinking before coding, you will know how software development can go wrong and therefore be looking for remedies like those offered in this book. This book is also suitable as a textbook in an advanced undergraduate or graduate level course.

Here is what to expect depending on what kind of reader you are:

Greenhorn developers or students. If you already have learned the basic mechanics of software development, such as programming languages and data structure design, and, ideally, have taken a general software engineering class, this book will introduce you to specific models of software that will help you form a *conceptual model* of software architecture. This model will help you make sense of the chaos of large systems without drawing a lot of diagrams and documentation. It may give you your first taste of ideas such as *quality attributes* and *architectural styles*. You will learn how to take your understanding of small programs and ramp it up to full industrial scale and quality. It can accelerate your progress toward becoming an effective, experienced developer.

Experienced developers. If you are good at developing systems then you will invariably be asked to mentor others. However, you may find that you have a somewhat idiosyncratic perspective on architecture, perhaps using unique diagram notations. This book will help you improve your ability to mentor others, understand why you are able to succeed where others struggle, and teach you about standard models, notations, and names.

Software architects. The role of software architect can be a difficult one when others in your organization do not understand what you do and why you do it. Not only will this book teach you techniques for building systems, it will also give you ways to explain what you are doing and how you are doing it. Perhaps you will even hand this book to co-workers so that you can better work as teammates.

Academics. This book makes several contributions to the field of software architecture. It introduces the *risk-driven model* of software architecture, which is a way of deciding how much architecture and design work to do on a project. It describes three approaches to architecture: *architecture-indifferent design*, *architecture-focused design*, and *architecture hoisting*. It integrates the functional camp's perspective on architecture with the quality-attribute camp's, yielding a single conceptual model. And it introduces the idea of an *architecturally-evident coding style* that makes your architecture evident from reading the source code.

Acknowledgments

This book would not have been possible without the generous assistance of many people. Several worked closely with me on one or more chapters and deserve special recognition for their help: Kevin Bierhoff, Alan Birchenough, David Garlan, Greg Hartman, Ian Maung, Paulo Merson, Bradley Schmerl, and Morgan Stanfield. Others suffered through bad early drafts, caught huge numbers of problems, and provided needed guidance: Len Bass, Grady Booch, Christopher Dean, Michael Donohue, Dan Dvorak, Anthony Earl, Hans Gyllstrom, Tim Halloran, Ralph Hoop, Michael Keeling, Ken LaToza, Thomas LaToza, Louis Marbel, Andy Myers, Carl Paradis, Paul Rayner, Patrick Riley, Aamod Sane, Nicholas Sherman, Olaf Zimmermann, and Guido Zgraggen. Thank you.

I would be remiss if I did not acknowledge all the people who have mentored me over the years, starting with my parents who provided more support than I can describe. My professional mentors have included Desmond D'Souza and the gang from Icon Computing; my thesis advisors, David Garlan and Bill Scherlis; and the faculty and students at Carnegie Mellon.

The wonderful cover illustration was concieved and drawn by my friend Lisa Haney (http://LisaHaney.com). Alan Apt has been a source of guidance and support through the book writing process.

The preparation of this book was done primarily with open source tools, including the Linux operating system, the LyX document processor, the Memoir LaTeX style, the LaTeX document preparation system, the Inkscape drawing editor, and Pavel Hruby's Visio UML template.

Contents