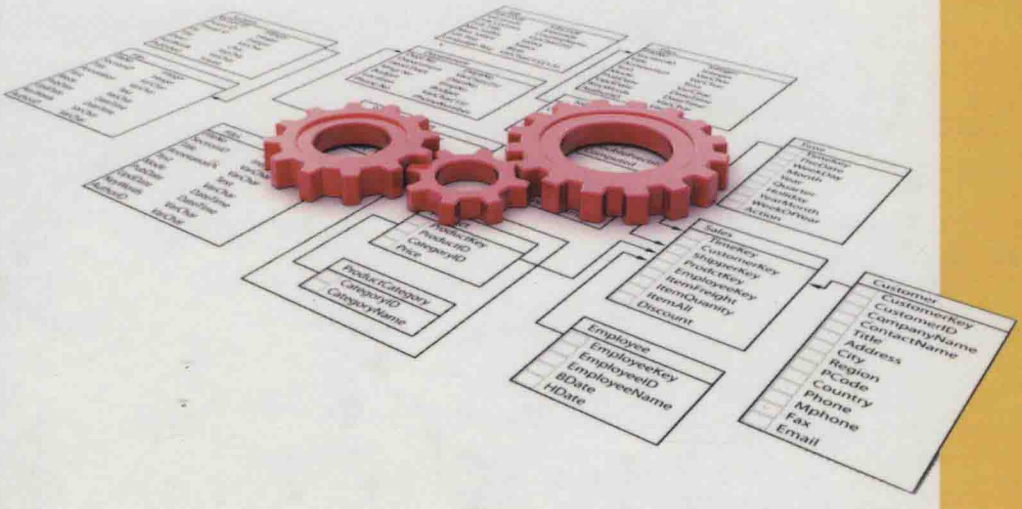
Software and Systems Architecture in Action



Raghvinder S. Sangwan

CRC Press
Taylor & Francis Group
AN AUERBACH BOOK

Software and Systems Architecture in Action

Raghvinder S. Sangwan

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business
AN AUERBACH BOOK

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

Printed on acid-free paper
Version Date: 20140418

International Standard Book Number-13: 978-1-4398-4916-3 (Hardback)

Library of Congress Cataloging-in-Publication Data

Sangwan, Raghvinder S.
Software and systems architecture in action / Raghvinder S. Sangwan.
pages cm. -- (Auerbach series on applied software engineering)
"A CRC title."
Includes bibliographical references and index.
ISBN 978-1-4398-4916-3
1. Computer architecture. 2. Software architecture. 3. Computer systems--Design and construction. 4. Electronic data processing--Distributed processing. 5. Business enterprise--Computer networks--Design and construction. I. Title.

QA76.9.A73S29 2015
004.2'2--dc23 2014013462

Visit the Taylor & Francis Web site at
http://www.taylorandfrancis.com

and the CRC Press Web site at
http://www.crcpress.com

Software and Systems Architecture in Action

Titles in the

Auerbach Series on Applied Software Engineering

Phillip A. Laplante, Pennsylvania State University, Series Editor

Software and Systems Architecture in Action
Raghvinder Sangwan
978-1-4398-4916-3

Requirements Engineering for Software and Systems, Second Edition
Phillip A. Laplante
978-1-4665-6081-9

Software Engineering Design: Theory and Practice
Carlos E. Otero
978-1-4398-5168-5

Ethics in IT Outsourcing
Tandy Gold
978-1-4398-5062-6

The ScrumMaster Study Guide
James Schiel
978-1-4398-5991-9

Antipatterns: Managing Software Organizations and People, Second Edition
Colin J. Neill, Philip A. Laplante, and Joanna F. DeFranco
978-1-4398-6186-8

Enterprise-Scale Agile Software Development
James Schiel
978-1-4398-0321-9

Building Software: A Practioner's Guide
Nikhilesh Krishnamurthy and Amitabh Saran
978-0-8493-7303-9

Global Software Development Handbook
Raghvinder Sangwan, Matthew Bass, Neel Mullick, Daniel J. Paulish, and Juergen Kazmeier
978-0-8493-9384-6

Software Engineering Quality Practices
Ronald Kirk Kandt
978-0-8493-4633-0

Other Auerbach Publications in Software Development, Software Engineering, and Project Management

Accelerating Process Improvement Using Agile Techniques
Deb Jacobs
0-8493-3796-8

The Complete Project Management Office Handbook
Gerard M. Hill
0-8493-2173-5

Defining and Deploying Software Processes
F. Alan Goodman
0-8493-9845-2

Embedded Linux System Design and Development
P. Raghavan, Amol Lad, and Sriram Neelakandan
0-8493-4058-6

Interpreting the CMMI®: A Process Improvement Approach
Margaret Kulpa and Kent Johnson
0-8493-1654-5

Modeling Software with Finite State Machines
Ferdinand Wagner, Ruedi Schmuki, Thomas Wagner, and Peter Wolstenholme
0-8493-8086-3

Optimizing Human Capital with a Strategic Project Office
J. Kent Crawford and Jeannette Cabanis-Brewin
0-8493-5410-2

A Practical Guide to Information Systems Strategic Planning, Second Edition
Anita Cassidy
0-8493-5073-5

Process-Based Software Project Management
F. Alan Goodman
0-8493-9845-2

Project Management Maturity Model, Second Edition
J. Kent Crawford
0-8493-7945-8

Real Process Improvement Using the CMMI®
Michael West
0-8493-2109-3

Reducing Risk with Software Process Improvement
Louis Poulin
0-8493-3828-X

The ROI from Software Quality
Khaled El Emam
0-8493-3298-2

Software Sizing, Estimation, and Risk Management
Daniel D. Galorath and Michael W. Evans
0-8493-3593-0

Software Specification and Design: An Engineering Approach
John C. Munson
0-8493-1992-7

Software Testing and Continuous Quality Improvement, Second Edition
William E. Lewis
0-8493-2524-2

Strategic Software Engineering: An Interdisciplinary Approach
Fadi P. Deek, James A.M. McHugh, and Osama M. Eljabiri
0-8493-3939-1

Successful Packaged Software Implementation
Christine B. Tayntor
0-8493-3410-1

Preface

Modern-day projects require software and systems engineers to work together in realizing architectures of large and complex software-intensive systems. To date, the two have been using their own concepts, techniques, methods, and tools when it comes to requirements, design, testing, maintenance, and evolution of these architectures. This book looks at synergies between the disciplines of software and systems engineering and explores practices that can help software and systems engineers work together more effectively as a unified team.

The book illustrates an approach to architecture design that is driven from systemic quality attributes determined from both the business and the technical goals of the system rather than just its functional requirements. This ensures that the architecture of the final system clearly, and traceably, reflects the most important goals for the system. While superficially the most important goals of any system are its functions, in practicality it is the quality attribute requirements that have the greatest impact on a system's lifetime value because it is these requirements that determine how easily the system accepts future change and how well the system meets the reliability and security needs of its operators and owners. By making these requirements "first-class citizens," the architecture meets them first.

Furthermore, most quality attribute requirements are systemic properties: They are properties that the entire system must reflect rather than just one component or subsystem. They

therefore cannot be easily built into an existing architecture. In essence, these properties must be designed into the architecture from the beginning. The architecture-centric design approach illustrated in this book addresses this directly by utilizing analytically derived patterns and tactics for quality attributes that inform the architect's design choices and help shape the architecture of a given system.

The book is organized into eight chapters. Chapter 1 focuses on the importance of architecture in modern-day systems. The amount and complexity of software in these systems are on the rise. Avionics software in modern aircraft has tens of millions of lines of code. In fighter aircraft, this software controls 80% of what a pilot does. It is not atypical for a premium-class automobile today to contain close to 100 million lines of code. The same is true for chemical and nuclear power plants. A large proportion of software in these systems introduce design and operational complexity, making them high-risk systems. This chapter highlights the role of architecture in managing this complexity and a need for an architecture-centric engineering approach to designing complex systems that can be used consistently by software and systems engineers working on such projects.

Chapter 2 looks at the influence of business goals or mission objectives on the architecture of a system. Business goals correspond to quality attributes the end system must exhibit. When using two different versions of a system, you may find them functionally equivalent but may develop a preference for one over the other because of its quick response time, ease of use, ease of modification, or high reliability. Such characteristics of a system are called quality attributes and are the predominant forces that shape the architecture of a system. Understanding business goals and their implied quality concerns is therefore critical.

A system operates within a given context or an environment, and understanding a system's operational aspects within its environment is also extremely important. Chapter 3 looks

at the concept of operations or ConOps, a term used for the operational view of the system from the perspective of its users that gives a broad understanding of the capabilities a system must deliver to fulfill its mission objectives. An operational view helps clearly delineate where a system's boundary is, what elements in its external environment a system must interact with, and what those interactions are.

If one must wait for a system to be developed to determine if it will meet the quality expectations of its stakeholders, there is an inherent risk that it may not. Architectural restructuring to achieve the desired qualities at this stage may be extremely difficult and costly. It is much more desirable to predict the systemic properties of a system from its design so corrections can be made before the system is committed to development. Patterns are known solutions to recurring design problems and therefore have qualities that can help predict the systemic properties of the system that is built using them. In prescribing solutions to problems, patterns may use many design decisions. These design decisions are known as tactics. Patterns and tactics are topics described in detail in Chapter 4.

When designing a system, one often must consider several requirements that have a strong influence on its architecture. Many of these requirements frequently conflict with each other. For instance, while one requirement may need the system to be highly secure, another may need the system to have quick response time. Making a system secure may introduce authentication, authorization, and encryption mechanisms that introduce latency, thereby slowing the system. Chapter 5 explores an approach to creating an architecture that systematically addresses the architecturally significant requirements while also dealing with trade-off situations created by requirements that conflict with each other.

Architecture is an artifact that serves many diverse needs for many diverse stakeholders. For instance, project managers use it for organizing projects and distributing the work among development teams, teams use it as a blueprint for their development

work and for understanding how their work depends on those of others, and maintainers use it to understand the impact of change as the system evolves over time. Effectively communicating the architecture to meet the diverse needs of a broad set of stakeholders is the topic of discussion for Chapter 6.

Once the architecture has been designed, the development teams must then undertake detailed design and implementation of the individual components that make up the final product. What is detailed design for one, however, is architecture to another. It turns out that effort must be invested by the development teams to develop the internal architecture of these components as they create the final blueprint for implementation. The interplay between architectural work and the detailed design is explored in Chapter 7.

Complexity is the topic of the final chapter, Chapter 8, which shows how following architecture-centric practices outlined in this text can lead to significant reduction in accidental complexity that is a by-product of development methodologies that lack focus on systemic properties of a system that have a strong influence on its architecture.

The fundamental objective of the book is to explore and illustrate practices that can be helpful in the development of architectures of large-scale systems in which software is a major component. It should be particularly useful to those currently involved in such projects and who are looking at more effective ways to engage the software and systems engineers on their teams. The book can be also used as a source for an undergraduate or graduate-level course in software and systems architecture as it exposes the students to concepts and techniques used for creating and managing architectures of software-intensive systems.

About the Author

Raghvinder (Raghu) Sangwan is an associate professor of software engineering at Pennsylvania State University. His work involves design and development of software systems, their architecture, and automatic and semiautomatic approaches to assess their design and code quality. He has published several papers in these areas. Prior to joining the Pennsylvania State University, Raghu was a software architect at Siemens, where he worked on large-scale systems in the domains of health care, automation, transportation, and mining; many of these systems were developed by teams geographically distributed around the world. This experience resulted in his coauthoring the *Global Software Development Handbook* and co-organizing the first International Conference on Global Software Engineering (ICGSE 2006), sponsored by the Institute for Electrical and Electronics Engineers (IEEE). He also holds a visiting scientist appointment at the Software Engineering Institute at Carnegie Mellon University. He earned his PhD in computer and information sciences from Temple University and is a senior member of IEEE and the Association for Computing Machinery (ACM).

Contents