Large-Scale Software Architecture

Large-Scale Software Architecture

A Practical Guide using UML

Jeff Garland
CrystalClear Software Inc.

Richard Anthony
Object Computing Inc.

JOHN WILEY & SONS, LTD

Other Wiley Editorial Offices

John Wiley & Sons Inc., 111 River Street, Hoboken, NJ 07030, USA

Jossey-Bass, 989 Market Street, San Francisco, CA 94103-1741, USA

Wiley-VCH Verlag GmbH, Boschstr. 12, D-69469 Weinheim, Germany

John Wiley & Sons Australia Ltd, 33 Park Road, Milton, Queensland 4064, Australia

John Wiley & Sons (Asia) Pte Ltd, 2 Clementi Loop #02-01, Jin Xing Distripark, Singapore 129809

John Wiley & Sons Canada Ltd, 22 Worcester Road, Etobicoke, Ontario, Canada M9W 1L1

Library of Congress Cataloging-in-Publication Data

(to follow)

British Library Cataloguing in Publication Data

A catalogue record for this book is available from the British Library

ISBN 0 470 84849 9
Typeset in 10½/13pt Sabon by Keytec Typesetting, Bridport, Dorset
Printed and bound in Great Britain by Biddles Ltd, Guildford and Kings Lynn
This book is printed on acid-free paper responsibly manufactured from sustainable forestry
in which at least two trees are planted for each one used for paper production.

Contents

Preface

The purpose of this book is to describe practical representations and techniques for the development of large-scale software architectures. The goal is to enable other software architects, developers, and managers to become more effective as a direct result of our experiences on several large-scale software development projects. We describe the techniques and architectural representations we have utilized successfully.

This book is intended to be a **practical guide**. Our goal is to be brief. We cover only the essential information to guide software architects in defining the software architecture, providing pointers to further reading in lieu of detailed descriptions of this literature. Ideally, we can help software development teams avoid the common practice of capturing the architecture after the software has been developed instead of utilizing the architecture as a tool to guide the software development.

The Unified Modeling Language (UML) is used throughout this book. We reduce the myriad of UML constructs to a precious few that we have found to be most useful. Leveraging the recent IEEE 1471 standard for software intensive systems, we describe several architectural viewpoints that are helpful in the development and documentation of software architectures. After reading this book, you will understand these viewpoints and techniques that will improve the modeling of your system's software architecture.

The focus of this book will be the software architecture of **large-scale systems**. Typically, this means **enterprise** systems and large **distributed** systems. However, most of the viewpoints and techniques discussed here will

apply to **smaller projects** and embedded systems. A typical large-scale software project will include:

- Large quantities of source code (typically millions of lines)
- Large numbers of developers (potentially hundreds, often geographically distributed)
- High complexity of interaction between components
- Extensive use of off-the-shelf components
- Multiple programming languages
- Multiple persistence mechanisms (files, relational databases, object databases)
- Multiple hardware platforms
- Distribution of components over several hardware platforms
- High concurrency

Dealing with the complexity of large-scale systems can be a challenge for even the most experienced software designers and developers. Large software systems contain millions of elements, which interact to achieve the system functionality. The interaction of these elements is far from obvious, especially given the artifacts created for a typical software project. These artifacts are especially critical as new team members are added and at different phases of the project. These phases include development, integration, testing and maintenance of the system. Even more challenging, however, these elements must been understood and modified as the required functionality of the system evolves. To do this correctly requires an understanding of how the elements interact as well as the underlying principles of the design.

Unfortunately, humans are ill equipped to manage complexity. Human short-term memory can typically hold between five and nine items simultaneously. Communication among team members is critical to cooperation and yet often uses imprecise language that frequently creates miscommunication. Providing a shared language of discussion can greatly enhance communication. Recently software has begun to develop some of the complexity management methods similar to those utilized in other engineering domains. These include the UML, object-modeling techniques, Design Patterns, and use of pre-fabricated software components and frameworks.

Architecture-based development is often recommended as a technique for

dealing with the complexity of large-scale projects. However, there is still little agreement about how to develop and describe software architecture effectively. The agreement usually ends with the use of UML for design, although this is not universal either. The UML provides a huge set of constructs for describing the software architecture, and includes many extensibility features. However, this flexibility creates a large number of possibilities for software architecture representation. In addition, most of the books and articles on software architecture and UML do not address large-scale development. The literature typically doesn't provide guidelines on how to get started in the definition of the software architecture, and doesn't provide specific representations which convey appropriate information to the stakeholders in a software architecture. This book is an attempt to meet these needs, which are critical to the software architect and the software development team.

Some areas where this book will provide practical guidance include:

- Modeling of architectural constructs, including components, subsystems, dependencies, transactions, and interfaces
- Modeling of environmental elements, including processes, nodes, and physical databases
- Insight into useful techniques for development of software architectures
- Various software architecture development processes
- Roles and responsibilities of the software architect and the architecture team
- Traps and pitfalls of architecture development
- Utilization of reusable and off-the-shelf software frameworks and components
- Addressing non-functional requirements such as performance and maintainability

This book does not purport to describe the best or only way to represent software architecture. Some systems may require additional representations from the ones shown in this book, and others may require only a subset of those shown here. However, most software development projects could benefit from at least some of the techniques and architecture representations described here.

In this book, we stick closely to the UML without major extensions. In some cases, this results in some limitations in formality or model semantics. Regardless of these limitations, these viewpoints have helped us solve complex problems in large systems. Note that over the course of several projects, the views described within were upgraded to utilize the UML. In many cases, we were using ad-hoc notations before the UML had reached its current state. In addition, future changes to the UML and the associated profiles may allow for improvements of the architecture views described in the book. Any that we are aware of are highlighted. Finally, although the focus is on modeling architecture with the UML, we discuss other representations where appropriate.

While a major portion of the book focuses on the application of the UML to software architecture, we also discuss the **role of the software architect** and how architecture development fits within the software development process. We have applied the architectural viewpoints described within on several projects across different organizations and within different development processes. Large projects tend to utilize relatively formal processes for which the described viewpoints fit nicely. However, we have also used these viewpoints and techniques on projects using highly iterative and **agile processes**. We believe that architecture-based development does not need to imply heavyweight processes.

The **intended audience** for this book includes those practitioners who are currently in the role of software architect, those who are currently software developers or designers and who will soon be in this role, and developers working on large-scale software development who want to better understand successful techniques for software architecture. We have assumed the reader has a working knowledge of the UML and at least a few years experience as a software developer or designer. Experience in the role of software architect or on a software architecture team would allow the reader to gain even more from reading this book.

This **book is organized** to provide general information and overview in the first chapters and discussion of specific architectural viewpoints in the later chapters. Chapter 1 provides our view of what 'software architecture' means. Chapters 2–3 discuss roles and process related to architecture. Chapter 4 gives an overview of a banking system example we use to illustrate the various viewpoints in the later chapters. Chapter 5 summarizes the UML diagrams and the viewpoints described in later chapters. Chapters 6–10 discuss and describe the various viewpoints of software architecture. Chapter 11 describes architecture development techniques and principles.

At the end of each chapter is a **recommended reading** list of key books and

papers. These references contain additional information on the topics covered in that chapter. Many of the books, papers, and URLs in the recommended readings provide detail in areas where we only touch lightly. This list is intended to contain the information we have found most useful. The books and papers are summarized in the Bibliography. URLs can be found at the book's web site.

Chapter 1 introduces the **definition of software architecture** and other terms. In addition, the UML-based architecture viewpoints are introduced and compared with other contemporary architecture methods.

Chapter 2 describes the **role of the software architect**. This includes topics such as the skills and background required to be an effective software architect, the ways an organization can support the architect, and the organization and structure of the software architecture team.

Chapter 3 discusses how software architecture relates to the overall **software development process** and describes processes for the development of software architecture. Topics include the creation of an effective review process, development of software infrastructure, technology roadmap management, process traps and pitfalls, and a brief discussion on tools.

Chapter 4 gives an overview of the **banking system example** that will be used to illustrate the architectural viewpoints described in the remainder of the book.

Chapter 5 provides a quick overview of the **UML diagrams and concepts** used in later chapters to build architectural viewpoints.

Chapter 6 provides an overview of representations and techniques for defining system context and performing **domain analysis**. Included is a discussion of conceptual diagrams, context views, and views used for domain analysis.

Chapter 7 explains architecture representations to facilitate **component development**. This includes the Component View, Component Interaction View, and Component State Views. Component messaging and interfaces are also discussed.

Chapter 8 discusses **subsystem and layer representations**. These views include the Layered Subsystem View and the Subsystem Interface Dependency View. These views serve as some of the fundamental diagrams utilized for software architecture.

Chapter 9 describes **transaction and logical data modeling**. This includes a discussion of mapping designs to relational databases.

Chapter 10 discusses representations for the modeling of **physical system** constructs, including nodes, databases, and process. These include Physical Data Views, Process Views, and Deployment Views.

Chapter 11 describes various **tips and techniques** essential to the development of software architectures. This includes architectural patterns, system partitioning, legacy and COTS utilization, and design techniques.

Chapter 12 puts it all together and has some **final remarks**. This includes some thoughts on becoming a software architect.

The Appendix provides **summaries** of all the **architectural viewpoints**.

This book provides a useful addition to the growing set of literature on software architecture in that it is a concise collection of key information, it is focused on large-scale software architecture, and it provides a set of key informative architectural viewpoints utilizing UML. We hope you will enjoy this book and find it to contain much of the key information required by the software architect. We welcome comments and discussion on this book at our website, *http://www.largescalesoftwarearchitecture.info/*.

Jeff Garland

Richard Anthony

Acknowledgments

We would like to thank the many individuals that helped make this book possible.

Our reviewers gave up their own valuable time to provide us very useful input that helped us to improve the overall quality of the book. These reviewers included Brad Appleton, Thomas Bichler, David DeLano, Robert Hanmer, Ralph Johnson (and his Software Architecture Class Participants), Patrick McDaid, Robert Nord, Micki Tugenberg, and Eoin Woods.

Thanks also to Linda Rising for inspiring us and helping us get this project started. In addition, we would like to thank our editor, Gaynor Redvers-Mutton. Without her enthusiasm and support, this book would not have been possible.

Finally, we would like to thank our families for their patience and support.