Large-Scale
Software Architecture
Large-Scale Software Architecture

A Practical Guide using UML

Jeff Garland
CrystalClear Software Inc.

Richard Anthony
Object Computing Inc.
## Contents

### Preface xi

### Acknowledgments xvii

### 1 Introduction 1

1.1 What is Software Architecture 1

1.1.1 What software architecture is not 4

1.1.2 Attributes of software architecture 5

1.1.3 Definitions of other key architecture-related terms 7

1.1.4 Other types of architectures 8

1.2 Why Architect? 10

1.3 Architectural Viewpoint Summary 12

1.4 Other Software Architecture Approaches 16

1.4.1 The 4+1 Views 16

1.4.2 RM-ODP viewpoints 17

1.4.3 Bass architectural structures 19

1.4.4 Hofmeister software architecture views 20

1.5 Recommended Reading 20

### 2 Roles of the Software Architect 21

2.1 Relationship to other key roles in development organization 25

Role: project management 25

Role: development team managers 25

Role: system architect/chief engineer 26

Role: chief software engineer 26

Role: hardware architect 27

Role: network architect 27
### 2. Skills and Background for the Architect

#### 2.2 Role: technical leads of each release
- data architect
- systems engineering leads
- software systems engineering lead

#### 2.3 Role: data architect

#### 2.4 Role: systems engineering leads

#### 2.5 Role: software systems engineering lead

### 2.5 Traps and Pitfalls Associated with the Role of Software Architect

#### 2.5.1 Clear definition of leadership

#### 2.5.2 Reporting structure for the software architect

#### 2.5.3 Geographical location of software architect and technical leads

#### 2.5.4 Architecture team size and composition

#### 2.5.5 Software architect lifecycle participation

### 2.6 Recommended Reading

### 3. Software Architecture and the Development Process

#### 3.1 Overview of Iterative Development

#### 3.1.1 Overall process phases

#### 3.1.2 Lifecycle stages

#### 3.1.3 Architecture and agile processes

#### 3.1.4 Start early, refine constantly

#### 3.2 Requirements Management

#### 3.2.1 Use cases and requirements engineering

#### 3.2.2 Additional requirements that impact architecture

#### 3.2.3 Requirements tracing

#### 3.3 Management of the Technology Roadmap

#### 3.3.1 External software products

#### 3.3.2 Software technology management traps and pitfalls

#### 3.3.3 Organizational technology roadmap

#### 3.4 Effective Technical Meetings

#### 3.4.1 Informal technical meetings

#### 3.4.2 Peer reviews and inspections

#### 3.4.3 Design reviews

#### 3.4.4 Design communication meetings

#### 3.4.5 Management meetings

#### 3.4.6 Vendor presentations

#### 3.4.7 Distributed technical meetings

#### 3.5 Traps and Pitfalls of the Software Architecture Process Activities

- The out-of-touch architect
- Analysis paralysis
- Design for reuse
- Use cases
- Schedule

#### 3.6 Computer-Aided Software Engineering (CASE) Tools

#### 3.7 Recommended Reading
## Contents

### 4 Example System Overview
- 4.1 System Overview 64
- 4.2 Overview of System Interfaces 64
- 4.3 Constraints 67
- 4.4 Major Operational Requirements and Software Requirements 67

### 5 UML Quick Tour
- 5.1 UML Diagram Summary 69
- 5.2 General Diagramming Conventions 72
- 5.2.1 General UML features: stereotypes, tagged values, multi-instance 73
- 5.2.2 View labels 74
- 5.3 The Diagrams 75
- 5.3.1 Component instance diagrams 75
- 5.3.2 Class and subsystem diagrams 76
- 5.3.3 Interaction (sequence and collaboration) diagrams 77
- 5.3.4 Deployment diagrams 79
- 5.3.5 Statechart diagrams 80
- 5.3.6 Activity diagrams 81
- 5.4 Managing Complexity 81
- 5.4.1 Use Case focused modeling 82
- 5.4.2 Element focused modeling 82
- 5.4.3 Level of detail 83
- 5.4.4 Controlling the number of models 83
- 5.4.5 Use Supplemental Textural Information 85
- 5.5 Recommended Reading 85

### 6 System Context and Domain Analysis
- 6.1 Conceptual Diagrams 87
- 6.2 Context Viewpoint 89
- 6.3 Domain Analysis Techniques 94
- 6.3.1 A formal analysis technique 95
- 6.3.2 Other techniques for finding domain entities 98
- 6.3.3 Analysis shortcuts 100
- 6.4 Analysis Viewpoints 101
- 6.4.1 Analysis Interaction Viewpoint 101
- 6.4.2 Analysis Focused Viewpoint 103
- 6.4.3 Analysis Overall Viewpoint 105
- 6.4.4 Candidate subsystem identification 107
- 6.5 Recommended Reading 108

### 7 Component Design and Modeling
- 7.1 Overview 111
- 7.1.1 Component-based development 111
- 7.1.2 Terminology 112
- 7.1.3 Communication and interfaces 115
- 7.1.4 Finding components 115
- 7.1.5 Qualities of component design 116
# Contents

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.2</td>
<td>Component Viewpoint</td>
<td>116</td>
</tr>
<tr>
<td>7.2.1</td>
<td>Component communication</td>
<td>117</td>
</tr>
<tr>
<td>7.2.2</td>
<td>Component interfaces</td>
<td>118</td>
</tr>
<tr>
<td>7.2.3</td>
<td>Message-based component modeling</td>
<td>121</td>
</tr>
<tr>
<td>7.2.4</td>
<td>Combining interfaces and messaging</td>
<td>124</td>
</tr>
<tr>
<td>7.2.5</td>
<td>Comparison of interfaces and messaging</td>
<td>127</td>
</tr>
<tr>
<td>7.2.6</td>
<td>Mechanism and performance annotations</td>
<td>128</td>
</tr>
<tr>
<td>7.3</td>
<td>Component Interaction Viewpoint</td>
<td>131</td>
</tr>
<tr>
<td>7.3.1</td>
<td>Component to Component Interactions</td>
<td>131</td>
</tr>
<tr>
<td>7.4</td>
<td>Component State Modeling</td>
<td>133</td>
</tr>
<tr>
<td>7.5</td>
<td>Modeling Highly Configurable Component Architectures</td>
<td>137</td>
</tr>
<tr>
<td>7.6</td>
<td>Recommended Reading</td>
<td>137</td>
</tr>
<tr>
<td>8</td>
<td>Subsystem Design</td>
<td>139</td>
</tr>
<tr>
<td>8.1</td>
<td>Terminology</td>
<td>139</td>
</tr>
<tr>
<td>8.2</td>
<td>Modeling Subsystems, Interfaces, and Layers</td>
<td>141</td>
</tr>
<tr>
<td>8.2.1</td>
<td>Subsystem Interface Dependency Viewpoint</td>
<td>141</td>
</tr>
<tr>
<td>8.2.2</td>
<td>Enhancing the Subsystem Dependency Views with layers</td>
<td>143</td>
</tr>
<tr>
<td>8.2.3</td>
<td>Top-level Dependencies</td>
<td>144</td>
</tr>
<tr>
<td>8.2.4</td>
<td>The Layered Subsystem Viewpoint</td>
<td>146</td>
</tr>
<tr>
<td>8.3</td>
<td>Mapping Subsystems and Layers to Implementation</td>
<td>151</td>
</tr>
<tr>
<td>8.3.1</td>
<td>Subsystems, layers, and build trees</td>
<td>151</td>
</tr>
<tr>
<td>8.3.2</td>
<td>Subsystems and components</td>
<td>153</td>
</tr>
<tr>
<td>8.4</td>
<td>Recommended Reading</td>
<td>154</td>
</tr>
<tr>
<td>9</td>
<td>Transaction and Data Design</td>
<td>155</td>
</tr>
<tr>
<td>9.1</td>
<td>Logical Data Architecture</td>
<td>155</td>
</tr>
<tr>
<td>9.1.1</td>
<td>Logical data model stability</td>
<td>157</td>
</tr>
<tr>
<td>9.1.2</td>
<td>Effects of the stable logical data model</td>
<td>158</td>
</tr>
<tr>
<td>9.2</td>
<td>Logical Data Viewpoint</td>
<td>159</td>
</tr>
<tr>
<td>9.2.1</td>
<td>Logical Data View example</td>
<td>160</td>
</tr>
<tr>
<td>9.2.2</td>
<td>Logical Data View for messaging</td>
<td>164</td>
</tr>
<tr>
<td>9.3</td>
<td>Data Model Design – Other Considerations</td>
<td>164</td>
</tr>
<tr>
<td>9.3.1</td>
<td>Data models and layers</td>
<td>164</td>
</tr>
<tr>
<td>9.3.2</td>
<td>Data models and reflection</td>
<td>165</td>
</tr>
<tr>
<td>9.3.3</td>
<td>Mapping objects to relational database</td>
<td>166</td>
</tr>
<tr>
<td>9.4</td>
<td>Transaction Design</td>
<td>169</td>
</tr>
<tr>
<td>9.4.1</td>
<td>Transaction concepts</td>
<td>170</td>
</tr>
<tr>
<td>9.4.2</td>
<td>Modeling transaction dynamics</td>
<td>171</td>
</tr>
<tr>
<td>9.4.3</td>
<td>Transactions and interface design</td>
<td>173</td>
</tr>
<tr>
<td>9.5</td>
<td>Recommended Reading</td>
<td>174</td>
</tr>
<tr>
<td>10</td>
<td>Process and Deployment Design</td>
<td>177</td>
</tr>
<tr>
<td>10.1</td>
<td>Physical Data Viewpoint</td>
<td>178</td>
</tr>
<tr>
<td>10.1.1</td>
<td>Modeling other storage attributes</td>
<td>179</td>
</tr>
<tr>
<td>10.1.2</td>
<td>Detailed physical storage modeling</td>
<td>181</td>
</tr>
<tr>
<td>10.2</td>
<td>Process Viewpoint</td>
<td>183</td>
</tr>
</tbody>
</table>
11 Architecture Techniques

11.1 Architecture Development Techniques
11.1.1 Commonality and variability analysis
11.1.2 Design for change
11.1.3 Generative programming techniques
11.1.4 Building a skeleton system
11.1.5 Prototyping
11.1.6 Interface development – Design by Contract
11.1.7 Architectural description languages
11.1.8 Architecture evaluation

11.2 Software Partitioning Strategies – Separation of Concerns
11.2.1 Functional decomposition
11.2.2 Isolate configuration data
11.2.3 Isolate hardware-specific components
11.2.4 Isolate time-critical components
11.2.5 Separate domain implementation model from human interface
11.2.6 Separate domain implementation model from implementation technology
11.2.7 Separate main function from monitoring
11.2.8 Separate fault recovery processing
11.2.9 Adaptation of external interfaces

11.3 Software Changeability and Dependency Management
11.3.1 The stable dependencies principle (SDP)
11.3.2 Acyclic dependencies principle
11.3.3 Interface Separation Principle

11.4 Using Architectural Patterns
11.5 Integration Strategies
11.5.1 Data-only integration
11.5.2 Executable integration

11.6 Establishing Architecture to Support Development
11.6.1 Configuration and change management
11.6.2 Build management
11.6.3 Continuous integration
11.6.4 Anticipate multi-language development
11.6.5 Anticipate tactical development (scripting)

11.7 Recommended Reading

12 Applying the Viewpoints

12.1 Bottom-Up Architecture Development
12.2 Top-Down Architecture Development
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.