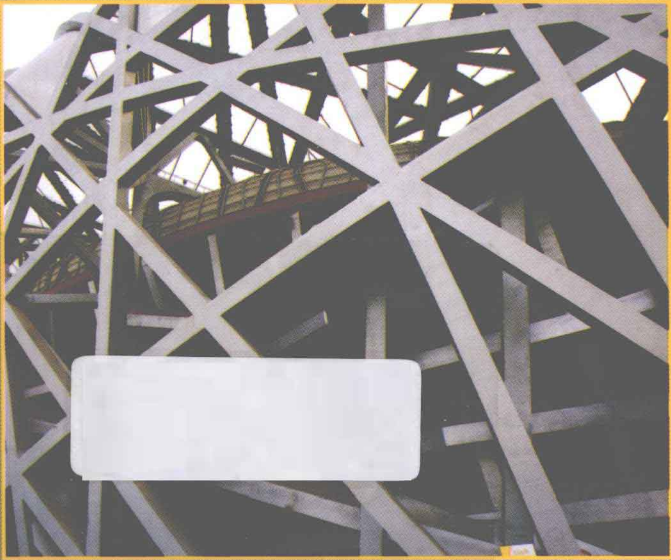
PEARSON

软件构架实践

（第3版 影印版）

Len Bass
Paul Clements 著
Rick Kazman

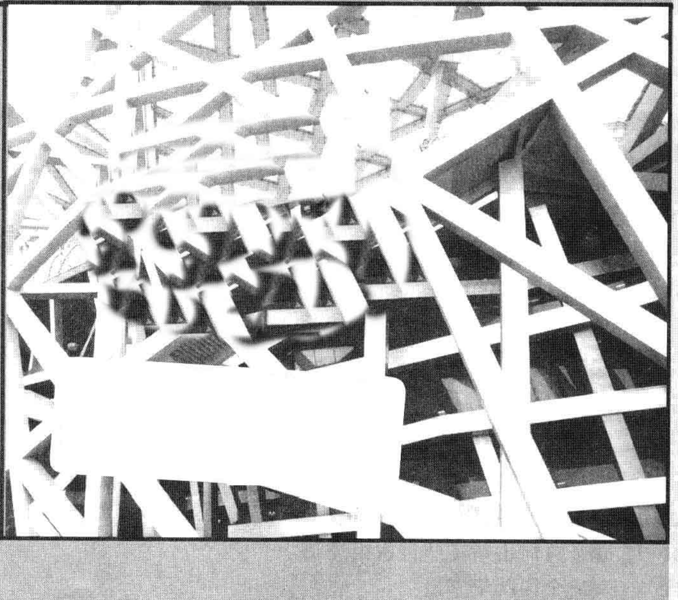


清华大学出版社

软件构架实践

（第3版 影印版）

Len Bass
Paul Clements
Rick Kazman 著



清华大学出版社
北 京

Original edition, entitled SOFTWARE ARCHITECTURE IN PRACTICE, 3E, 9780321815736 by BASS, LEN; CLEMENTS, PAUL; KAZMAN, RICK, published by Pearson Education, Inc, publishing as Addison-Wesley Professional, Copyright © 2013 Pearson Education, Inc..
All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc..
China edition published by PEARSON EDUCATION ASIA LTD., and TSINGHUA UNIVERSITY PRESS LIMITED Copyright © 2013.
This edition is manufactured in the People's Republic of China, and is authorized for sale and distribution in the People's Republic of China exclusively (except Taiwan, Hong Kong SAR and Macau SAR).
仅限于中华人民共和国境内(不包括中国香港、澳门特别行政区和中国台湾地区)销售发行。

本书封面贴有 Pearson Education(培生教育出版集团)激光防伪标签，无标签者不得销售。
版权所有，侵权必究。侵权举报电话：010-62782989　13701121933
北京市版权局著作权合同登记号　图字：01-2012-8971

图书在版编目(CIP)数据

软件构架实践第 3 版 = Software Architecture in Practice：英文/(美)巴斯(Bass, L.)，(美)克莱门茨(Clements, P.)，(美)凯兹曼(Kazman, R.)著. --影印本. —北京：清华大学出版社，2013
ISBN 978-7-302-31293-2

Ⅰ. ①软…　Ⅱ. ①巴…　②克…　③凯…　Ⅲ. ①软件开发—英文　Ⅳ. ①TP311.52

中国版本图书馆 CIP 数据核字(2013)第 013981 号

责任编辑：文开琪
封面设计：杨玉兰
责任校对：周剑云
责任印制：李红英

出版发行：清华大学出版社
网　址：http://www.tup.com.cn，http://www.wqbook.com
地　址：北京清华大学学研大厦 A 座　邮　编：100084
社总机：010-62770175　邮　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课 件 下 载：http://www.tup.com.cn，010-62791865
印 刷 者：三河市君旺印装厂
装 订 者：三河市新茂装订有限公司
经　销：全国新华书店
开　本：159mm×236mm　印　张：38.75　字　数：790 千字
版　次：2013 年 2 月第 3 版　印　次：2013 年 2 月第1 次印刷
印　数：1～4000
定　价：79.00 元

产品编号：050690-01

Contents

Preface *xv*

Reader's Guide *xvii*

Acknowledgments *xix*

PART ONE INTRODUCTION 1

CHAPTER 1 What Is Software Architecture? 3

1.1 What Software Architecture Is and What It Isn't 4

1.2 Architectural Structures and Views 9

1.3 Architectural Patterns 18

1.4 What Makes a "Good" Architecture? 19

1.5 Summary 21

1.6 For Further Reading 22

1.7 Discussion Questions 23

CHAPTER 2 Why Is Software Architecture Important? 25

2.1 Inhibiting or Enabling a System's Quality Attributes 26

2.2 Reasoning About and Managing Change 27

2.3 Predicting System Qualities 28

2.4 Enhancing Communication among Stakeholders 29

2.5 Carrying Early Design Decisions 31

2.6 Defining Constraints on an Implementation 32

2.7 Influencing the Organizational Structure 33

2.8 Enabling Evolutionary Prototyping 33

2.9 Improving Cost and Schedule Estimates 34
2.10 Supplying a Transferable, Reusable Model 35
2.11 Allowing Incorporation of Independently Developed Components 35
2.12 Restricting the Vocabulary of Design Alternatives 36
2.13 Providing a Basis for Training 37
2.14 Summary 37
2.15 For Further Reading 38
2.16 Discussion Questions 38

CHAPTER 3 The Many Contexts of Software Architecture 39

3.1 Architecture in a Technical Context 40
3.2 Architecture in a Project Life-Cycle Context 44
3.3 Architecture in a Business Context 49
3.4 Architecture in a Professional Context 51
3.5 Stakeholders 52
3.6 How Is Architecture Influenced? 56
3.7 What Do Architectures Influence? 57
3.8 Summary 59
3.9 For Further Reading 59
3.10 Discussion Questions 60

PART TWO QUALITY ATTRIBUTES 61

CHAPTER 4 Understanding Quality Attributes 63

4.1 Architecture and Requirements 64
4.2 Functionality 65
4.3 Quality Attribute Considerations 65
4.4 Specifying Quality Attribute Requirements 68
4.5 Achieving Quality Attributes through Tactics 70
4.6 Guiding Quality Design Decisions 72
4.7 Summary 76

试读结束，需要全本PDF请购买 www.ertongbook.com

4.8 For Further Reading 77
4.9 Discussion Questions 77

CHAPTER 5 Availability 79
5.1 Availability General Scenario 85
5.2 Tactics for Availability 87
5.3 A Design Checklist for Availability 96
5.4 Summary 98
5.5 For Further Reading 99
5.6 Discussion Questions 100

CHAPTER 6 Interoperability 103
6.1 Interoperability General Scenario 107
6.2 Tactics for Interoperability 110
6.3 A Design Checklist for Interoperability 114
6.4 Summary 115
6.5 For Further Reading 116
6.6 Discussion Questions 116

CHAPTER 7 Modifiability 117
7.1 Modifiability General Scenario 119
7.2 Tactics for Modifiability 121
7.3 A Design Checklist for Modifiability 125
7.4 Summary 128
7.5 For Further Reading 128
7.6 Discussion Questions 128

CHAPTER 8 Performance 131
8.1 Performance General Scenario 132
8.2 Tactics for Performance 135
8.3 A Design Checklist for Performance 142
8.4 Summary 145
8.5 For Further Reading 145
8.6 Discussion Questions 145

CHAPTER 9 Security 147
9.1 Security General Scenario 148
9.2 Tactics for Security 150

9.3 A Design Checklist for Security 154
9.4 Summary 156
9.5 For Further Reading 157
9.6 Discussion Questions 158

CHAPTER 10 Testability 159
10.1 Testability General Scenario 162
10.2 Tactics for Testability 164
10.3 A Design Checklist for Testability 169
10.4 Summary 172
10.5 For Further Reading 172
10.6 Discussion Questions 173

CHAPTER 11 Usability 175
11.1 Usability General Scenario 176
11.2 Tactics for Usability 177
11.3 A Design Checklist for Usability 181
11.4 Summary 183
11.5 For Further Reading 183
11.6 Discussion Questions 183

CHAPTER 12 Other Quality Attributes 185
12.1 Other Important Quality Attributes 185
12.2 Other Categories of Quality Attributes 189
12.3 Software Quality Attributes and System Quality Attributes 190
12.4 Using Standard Lists of Quality Attributes—or Not 193
12.5 Dealing with "X-ability": Bringing a New Quality Attribute into the Fold 196
12.6 For Further Reading 200
12.7 Discussion Questions 201

CHAPTER 13 Architectural Tactics and Patterns 203
13.1 Architectural Patterns 204
13.2 Overview of the Patterns Catalog 205
13.3 Relationships between Tactics and Patterns 238

13.4 Using Tactics Together 242
13.5 Summary 247
13.6 For Further Reading 248
13.7 Discussion Questions 249

CHAPTER 14 Quality Attribute Modeling and Analysis 251
14.1 Modeling Architectures to Enable Quality Attribute Analysis 252
14.2 Quality Attribute Checklists 260
14.3 Thought Experiments and Back-of-the-Envelope Analysis 262
14.4 Experiments, Simulations, and Prototypes 264
14.5 Analysis at Different Stages of the Life Cycle 265
14.6 Summary 266
14.7 For Further Reading 267
14.8 Discussion Questions 269

PART THREE ARCHITECTURE IN THE LIFE CYCLE 271

CHAPTER 15 Architecture in Agile Projects 275
15.1 How Much Architecture? 277
15.2 Agility and Architecture Methods 281
15.3 A Brief Example of Agile Architecting 283
15.4 Guidelines for the Agile Architect 286
15.5 Summary 287
15.6 For Further Reading 288
15.7 Discussion Questions 289

CHAPTER 16 Architecture and Requirements 291
16.1 Gathering ASRs from Requirements Documents 292
16.2 Gathering ASRs by Interviewing Stakeholders 294
16.3 Gathering ASRs by Understanding the Business Goals 296

16.4 Capturing ASRs in a Utility Tree 304
16.5 Tying the Methods Together 308
16.6 Summary 308
16.7 For Further Reading 309
16.8 Discussion Questions 309

CHAPTER 17 Designing an Architecture 311

17.1 Design Strategy 311
17.2 The Attribute-Driven Design Method 316
17.3 The Steps of ADD 318
17.4 Summary 325
17.5 For Further Reading 325
17.6 Discussion Questions 326

CHAPTER 18 Documenting Software Architectures 327

18.1 Uses and Audiences for Architecture Documentation 328
18.2 Notations for Architecture Documentation 329
18.3 Views 331
18.4 Choosing the Views 341
18.5 Combining Views 343
18.6 Building the Documentation Package 345
18.7 Documenting Behavior 351
18.8 Architecture Documentation and Quality Attributes 354
18.9 Documenting Architectures That Change Faster Than You Can Document Them 355
18.10 Documenting Architecture in an Agile Development Project 356
18.11 Summary 359
18.12 For Further Reading 360
18.13 Discussion Questions 360

CHAPTER 19 Architecture, Implementation, and Testing 363

19.1 Architecture and Implementation 363
19.2 Architecture and Testing 370

19.3 Summary 376
19.4 For Further Reading 376
19.5 Discussion Questions 377

CHAPTER 20 Architecture Reconstruction and Conformance 379

20.1 Architecture Reconstruction Process 381
20.2 Raw View Extraction 382
20.3 Database Construction 386
20.4 View Fusion 388
20.5 Architecture Analysis: Finding Violations 389
20.6 Guidelines 392
20.7 Summary 393
20.8 For Further Reading 394
20.9 Discussion Questions 395

CHAPTER 21 Architecture Evaluation 397

21.1 Evaluation Factors 397
21.2 The Architecture Tradeoff Analysis Method 400
21.3 Lightweight Architecture Evaluation 415
21.4 Summary 417
21.5 For Further Reading 417
21.6 Discussion Questions 418

CHAPTER 22 Management and Governance 419

22.1 Planning 420
22.2 Organizing 422
22.3 Implementing 427
22.4 Measuring 429
22.5 Governance 430
22.6 Summary 432
22.7 For Further Reading 432
22.8 Discussion Questions 433

PART FOUR ARCHITECTURE AND BUSINESS 435

CHAPTER 23 **Economic Analysis of Architectures** 437

23.1 Decision-Making Context 438
23.2 The Basis for the Economic Analyses 439
23.3 Putting Theory into Practice: The CBAM 442
23.4 Case Study: The NASA ECS Project 450
23.5 Summary 457
23.6 For Further Reading 458
23.7 Discussion Questions 458

CHAPTER 24 **Architecture Competence** 459

24.1 Competence of Individuals: Duties, Skills, and Knowledge of Architects 460
24.2 Competence of a Software Architecture Organization 467
24.3 Summary 475
24.4 For Further Reading 475
24.5 Discussion Questions 477

CHAPTER 25 **Architecture and Software Product Lines** 479

25.1 An Example of Product Line Variability 482
25.2 What Makes a Software Product Line Work? 483
25.3 Product Line Scope 486
25.4 The Quality Attribute of Variability 488
25.5 The Role of a Product Line Architecture 488
25.6 Variation Mechanisms 490
25.7 Evaluating a Product Line Architecture 493
25.8 Key Software Product Line Issues 494
25.9 Summary 497
25.10 For Further Reading 498
25.11 Discussion Questions 498

试读结束，需要全本PDF请购买 www.ertongbook.com

PART FIVE THE BRAVE NEW WORLD 501

CHAPTER 26 Architecture in the Cloud 503

26.1 Basic Cloud Definitions 504
26.2 Service Models and Deployment Options 505
26.3 Economic Justification 506
26.4 Base Mechanisms 509
26.5 Sample Technologies 514
26.6 Architecting in a Cloud Environment 520
26.7 Summary 524
26.8 For Further Reading 524
26.9 Discussion Questions 525

CHAPTER 27 Architectures for the Edge 527

27.1 The Ecosystem of Edge-Dominant Systems 528
27.2 Changes to the Software Development Life Cycle 530
27.3 Implications for Architecture 531
27.4 Implications of the Metropolis Model 533
27.5 Summary 537
27.6 For Further Reading 538
27.7 Discussion Questions 538

CHAPTER 28 Epilogue 541

References 547

About the Authors 561

Index 563

PART ONE

INTRODUCTION

What is a software architecture? What is it good for? How does it come to be? What effect does its existence have? These are the questions we answer in Part I.

Chapter 1 deals with a technical perspective on software architecture. We define it and relate it to system and enterprise architectures. We discuss how the architecture can be represented in different views to emphasize different perspectives on the architecture. We define patterns and discuss what makes a "good" architecture.

In Chapter 2, we discuss the uses of an architecture. You may be surprised that we can find so many—ranging from a vehicle for communication among stakeholders to a blueprint for implementation, to the carrier of the system's quality attributes. We also discuss how the architecture provides a reasoned basis for schedules and how it provides the foundation for training new members on a team.

Finally, in Chapter 3, we discuss the various contexts in which a software architecture exists. It exists in a technical context, in a project life-cycle context, in a business context, and in a professional context. Each of these contexts defines a role for the software architecture to play, or an influence on it. These impacts and influences define the Architecture Influence Cycle.

1

What Is Software Architecture?

Good judgment is usually the result of experience. And experience is frequently the result of bad judgment. But to learn from the experience of others requires those who have the experience to share the knowledge with those who follow.
—Barry LePatner

Writing (on our part) and reading (on your part) a book about software architecture, which distills the experience of many people, presupposes that

1. having a software architecture is important to the successful development of a software system and
2. there is a sufficient, and sufficiently generalizable, body of knowledge about software architecture to fill up a book.

One purpose of this book is to convince you that both of these assumptions are true, and once you are convinced, give you a basic knowledge so that you can apply it yourself.

Software systems are constructed to satisfy organizations' business goals. The architecture is a bridge between those (often abstract) business goals and the final (concrete) resulting system. While the path from abstract goals to concrete systems can be complex, the good news is that software architectures can be designed, analyzed, documented, and implemented using known techniques that will support the achievement of these business and mission goals. The complexity can be tamed, made tractable.

These, then, are the topics for this book: the design, analysis, documentation, and implementation of architectures. We will also examine the influences, principally in the form of business goals and quality attributes, which inform these activities.

In this chapter we will focus on architecture strictly from a software engineering point of view. That is, we will explore the value that a software architecture brings to a development project. (Later chapters will take a business and organizational perspective.)

1.1 What Software Architecture Is and What It Isn't

There are many definitions of software architecture, easily discoverable with a web search, but the one we like is this one:

> The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both.

This definition stands in contrast to other definitions that talk about the system's "early" or "major" design decisions. While it is true that many architectural decisions are made early, not all are—especially in Agile or spiral-development projects. It's also true that very many decisions are made early that are not architectural. Also, it's hard to look at a decision and tell whether or not it's "major." Sometimes only time will tell. And since writing down an architecture is one of the architect's most important obligations, we need to know now which decisions an architecture comprises.

Structures, on the other hand, are fairly easy to identify in software, and they form a powerful tool for system design.

Let us look at some of the implications of our definition.

Architecture Is a Set of Software Structures

This is the first and most obvious implication of our definition. A structure is simply a set of elements held together by a relation. Software systems are composed of many structures, and no single structure holds claim to being *the* architecture. There are three categories of architectural structures, which will play an important role in the design, documentation, and analysis of architectures:

1. First, some structures partition systems into implementation units, which in this book we call *modules*. Modules are assigned specific computational responsibilities, and are the basis of work assignments for programming teams (Team A works on the database, Team B works on the business rules, Team C works on the user interface, etc.). In large projects, these elements (modules) are subdivided for assignment to subteams. For example, the database for a large enterprise resource planning (ERP) implementation might be so complex that its implementation is split into many parts. The structure that captures that decomposition is a kind of module structure, the *module*

decomposition structure in fact. Another kind of module structure emerges as an output of object-oriented analysis and design—class diagrams. If you aggregate your modules into layers, you've created another (and very useful) module structure. Module structures are static structures, in that they focus on the way the system's functionality is divided up and assigned to implementation teams.

2. Other structures are dynamic, meaning that they focus on the way the elements interact with each other at runtime to carry out the system's functions. Suppose the system is to be built as a set of services. The services, the infrastructure they interact with, and the synchronization and interaction relations among them form another kind of structure often used to describe a system. These services are made up of (compiled from) the programs in the various implementation units that we just described. In this book we will call runtime structures *component-and-connector* (C&C) structures. The term *component* is overloaded in software engineering. In our use, a component is always a runtime entity.
3. A third kind of structure describes the mapping from software structures to the system's organizational, developmental, installation, and execution environments. For example, modules are assigned to teams to develop, and assigned to places in a file structure for implementation, integration, and testing. Components are deployed onto hardware in order to execute. These mappings are called *allocation* structures.

Although software comprises an endless supply of structures, not all of them are architectural. For example, the set of lines of source code that contain the letter "z," ordered by increasing length from shortest to longest, is a software structure. But it's not a very interesting one, nor is it architectural. A structure is architectural if it supports reasoning about the system and the system's properties. The reasoning should be about an attribute of the system that is important to some stakeholder. These include functionality achieved by the system, the availability of the system in the face of faults, the difficulty of making specific changes to the system, the responsiveness of the system to user requests, and many others. We will spend a great deal of time in this book on the relationship between architecture and quality attributes like these.

Thus, the set of architectural structures is not fixed or limited. What is architectural is what is useful in your context for your system.

Architecture Is an Abstraction

Because architecture consists of structures and structures consist of elements[1] and relations, it follows that an architecture comprises software elements and

1. In this book we use the term "element" when we mean either a module or a component, and don't want to distinguish.

试读结束，需要全本PDF请购买 www.ertongbook.com