

《软件工程》

图书基本信息

书名：《软件工程》

13位ISBN编号：9787111341963

10位ISBN编号：7111341961

出版时间：2011-6

出版社：机械工业出版社

作者：（美）Stephen R. Schach

页数：667

版权说明：本站所提供下载的PDF图书仅提供预览和简介以及在线试读，请支持正版图书。

更多资源请访问：www.tushu000.com

《软件工程》

内容概要

本书对软件工程的基础知识（包括面向对象和传统方法）进行了严谨和全面的介绍，是软件工程领域的经典著作。

全书共分两大部分：第一部分介绍基本的软件工程理论；第二部分讲述更实用的软件生命周期。作者采用这种独特的、极具可读性的组织方式，帮助学生和广大读者理解软件工程中的一些复杂概念。最新版第8版对全书进行了整体更新，新增两章内容，分别概括介绍软件工程的关键知识点和近年涌现的新技术。

本版新增内容如下：

新增两章内容：第10章总结第一部分涉及的关键知识点，便于学生在做团队项目时参考使用；第18章概括介绍10种新技术，分别是面向方面的技术、模型驱动技术、基于组件的技术、面向服务的技术、社交计算、Web工程、云技术、Web 3.0和模型检测。

扩展设计模式的相关材料，新增一个小的案例。

新增两个理论工具，即分而治之和关注点分离。

新增100多道习题，并更新了大量参考文献。

《软件工程》

作者简介

Stephen R. Schach 1972年获魏兹曼科学院物理学理科硕士学位，1973年获开普敦大学应用数学博士学位，目前是美国范德比尔特大学计算机科学和计算机工程名誉教授。他的研究兴趣主要集中在软件工程领域，特别是对软件维护与开源软件的实验分析有深入研究。他著有多部软件工程、面向对象系统分析与设计方面的教材。

书籍目录

Preface	iv
Chapter 1	
The Scope of Software Engineering	1
Learning Objectives	1
1.1 Historical Aspects	2
1.2 Economic Aspects	5
1.3 Maintenance Aspects	6
1.3.1 Classical and Modern Views of Maintenance	9
1.3.2 The Importance of Postdelivery Maintenance	10
1.4 Requirements, Analysis, and Design Aspects	12
1.5 Team Development Aspects	15
1.6 Why There Is No Planning Phase	16
1.7 Why There Is No Testing Phase	16
1.8 Why There Is No Documentation Phase	17
1.9 The Object-Oriented Paradigm	18
1.10 The Object-Oriented Paradigm in Perspective	22
1.11 Terminology	23
1.12 Ethical Issues	26
Chapter Review	27
For Further Reading	27
Key Terms	28
Problems	29
References	30
PART A	
SOFTWARE ENGINEERING CONCEPTS	35
Chapter 2	
Software Life-Cycle Models	37
Learning Objectives	37
2.1 Software Development in Theory	37
2.2 Winburg Mini Case Study	38
2.3 Lessons of the Winburg Mini Case Study	42
2.4 Teal Tractors Mini Case Study	42
2.5 Iteration and Incrementation	43
2.6 Winburg Mini Case Study Revisited	47
2.7 Risks and Other Aspects of Iteration and Incrementation	48
2.8 Managing Iteration and Incrementation	51
2.9 Other Life-Cycle Models	52
2.9.1 Code-and-Fix Life-Cycle Model	52
2.9.2 Waterfall Life-Cycle Model	53
2.9.3 Rapid-Prototyping Life-Cycle Model	55
2.9.4 Open-Source Life-Cycle Model	56
2.9.5 Agile Processes	59
2.9.6 Synchronize-and-Stabilize Life-Cycle Model	62
2.9.7 Spiral Life-Cycle Model	62
2.10 Comparison of Life-Cycle Models	66

Chapter Review	67
For Further Reading	68
Key Terms	69
Problems	69
References	70
Chapter 3	
The Software Process	74
Learning Objectives	74
3.1 The Unified Process	76
3.2 Iteration and Incrementation within the Object-Oriented Paradigm	76
3.3 The Requirements Workflow	78
3.4 The Analysis Workflow	80
3.5 The Design Workflow	82
3.6 The Implementation Workflow	83
3.7 The Test Workflow	84
3.7.1 Requirements Artifacts	84
3.7.2 Analysis Artifacts	84
3.7.3 Design Artifacts	85
3.7.4 Implementation Artifacts	85
3.8 Postdelivery Maintenance	87
3.9 Retirement	88
3.10 The Phases of the Unified Process	88
3.10.1 The Inception Phase	89
3.10.2 The Elaboration Phase	91
3.10.3 The Construction Phase	92
3.10.4 The Transition Phase	92
3.11 One- versus Two-Dimensional Life-Cycle Models	92
3.12 Improving the Software Process	94
3.13 Capability Maturity Models	95
3.14 Other Software Process Improvement Initiatives	98
3.15 Costs and Benefits of Software Process Improvement	99
Chapter Review	101
For Further Reading	102
Key Terms	102
Problems	103
References	104
Chapter 4	
Teams	107
Learning Objectives	107
4.1 Team Organization	107
4.2 Democratic Team Approach	109
4.2.1 Analysis of the Democratic Team Approach	110
4.3 Classical Chief Programmer Team Approach	110
4.3.1 The New York Times Project	112
4.3.2 Impracticality of the Classical Chief Programmer Team Approach	113
4.4 Beyond Chief Programmer and Democratic Teams	113
4.5 Synchronize-and-Stabilize Teams	117
4.6 Teams for Agile Processes	118

4.7 Open-Source Programming Teams	118
4.8 People Capability Maturity Model	119
4.9 Choosing an Appropriate Team Organization	120
Chapter Review	121
For Further Reading	121
Key Terms	122
Problems	122
References	122
Chapter 5	
The Tools of the Trade	124
Learning Objectives	124
5.1 Stepwise Refinement	124
5.1.1 Stepwise Refinement Mini Case Study	125
5.2 Cost – Benefit Analysis	130
5.3 Divide-and-Conquer	132
5.4 Separation of Concerns	132
5.5 Software Metrics	133
5.6 CASE	134
5.7 Taxonomy of CASE	135
5.8 Scope of CASE	137
5.9 Software Versions	141
5.9.1 Revisions	141
5.9.2 Variations	142
5.10 Configuration Control	143
5.10.1 Configuration Control during Postdelivery Maintenance	145
5.10.2 Baselines	145
5.10.3 Configuration Control during Development	146
5.11 Build Tools	146
5.12 Productivity Gains with CASE Technology	147
Chapter Review	149
For Further Reading	149
Key Terms	150
Problems	150
References	151
Chapter 6	
Testing	154
Learning Objectives	154
6.1 Quality Issues	155
6.1.1 Software Quality Assurance	156
6.1.2 Managerial Independence	156
6.2 Non-Execution-Based Testing	157
6.2.1 Walkthroughs	158
6.2.2 Managing Walkthroughs	158
6.2.3 Inspections	159
6.2.4 Comparison of Inspections and Walkthroughs	161
6.2.5 Strengths and Weaknesses of Reviews	162
6.2.6 Metrics for Inspections	162
6.3 Execution-Based Testing	162
6.4 What Should Be Tested?	163

6.4.1 Utility	164
6.4.2 Reliability	164
6.4.3 Robustness	165
6.4.4 Performance	165
6.4.5 Correctness	166
6.5 Testing versus Correctness Proofs	167
6.5.1 Example of a Correctness Proof	167
6.5.2 Correctness Proof Mini Case Study	171
6.5.3 Correctness Proofs and Software Engineering	172
6.6 Who Should Perform Execution-Based Testing?	175
6.7 When Testing Stops	176
Chapter Review	176
For Further Reading	177
Key Terms	177
Problems	178
References	179
Chapter 7	
From Modules to Objects	183
Learning Objectives	183
7.1 What Is a Module?	183
7.2 Cohesion	187
7.2.1 Coincidental Cohesion	187
7.2.2 Logical Cohesion	188
7.2.3 Temporal Cohesion	189
7.2.4 Procedural Cohesion	189
7.2.5 Communicational Cohesion	190
7.2.6 Functional Cohesion	190
7.2.7 Informational Cohesion	191
7.2.8 Cohesion Example	191
7.3 Coupling	192
7.3.1 Content Coupling	192
7.3.2 Common Coupling	193
7.3.3 Control Coupling	195
7.3.4 Stamp Coupling	195
7.3.5 Data Coupling	196
7.3.6 Coupling Example	197
7.3.7 The Importance of Coupling	198
7.4 Data Encapsulation	199
7.4.1 Data Encapsulation and Development	201
7.4.2 Data Encapsulation and Maintenance	202
7.5 Abstract Data Types	207
7.6 Information Hiding	209
7.7 Objects	211
7.8 Inheritance, Polymorphism, and Dynamic Binding	215
7.9 The Object-Oriented Paradigm	217
Chapter Review	220
For Further Reading	221
Key Terms	221
Problems	221

References	222
Chapter 8	
Reusability and Portability	225
Learning Objectives	225
8.1 Reuse Concepts	226
8.2 Impediments to Reuse	228
8.3 Reuse Case Studies	229
8.3.1 Raytheon Missile Systems Division	230
8.3.2 European Space Agency	231
8.4 Objects and Reuse	232
8.5 Reuse during Design and Implementation	232
8.5.1 Design Reuse	232
8.5.2 Application Frameworks	234
8.5.3 Design Patterns	235
8.5.4 Software Architecture	236
8.5.5 Component-Based Software Engineering	237
8.6 More on Design Patterns	237
8.6.1 FLIC Mini Case Study	238
8.6.2 Adapter Design Pattern	239
8.6.3 Bridge Design Pattern	240
8.6.4 Iterator Design Pattern	241
8.6.5 Abstract Factory Design Pattern	241
8.7 Categories of Design Patterns	245
8.8 Strengths and Weaknesses of Design Patterns	247
8.9 Reuse and the World Wide Web	248
8.10 Reuse and Postdelivery Maintenance	249
8.11 Portability	250
8.11.1 Hardware Incompatibilities	250
8.11.2 Operating System Incompatibilities	251
8.11.3 Numerical Software Incompatibilities	251
8.11.4 Compiler Incompatibilities	253
8.12 Why Portability?	255
8.13 Techniques for Achieving Portability	256
8.13.1 Portable System Software	257
8.13.2 Portable Application Software	257
8.13.3 Portable Data	258
8.13.4 Model-Driven Architecture	259
Chapter Review	259
For Further Reading	260
Key Terms	261
Problems	261
References	263
CHAPTER 9	
Planning and Estimating	268
Learning Objectives	268
9.1 Planning and the Software Process	268
9.2 Estimating Duration and Cost	270
9.2.1 Metrics for the Size of a Product	272

9.2.2	Techniques of Cost Estimation	275
9.2.3	Intermediate COCOMO	278
9.2.4	COCOMO II	281
9.2.5	Tracking Duration and Cost Estimates	282
9.3	Components of a Software Project Management Plan	282
9.4	Software Project Management Plan Framework	284
9.5	IEEE Software Project Management Plan	286
9.6	Planning Testing	288
9.7	Planning Object-Oriented Projects	289
9.8	Training Requirements	290
9.9	Documentation Standards	291
9.10	CASE Tools for Planning and Estimating	292
9.11	Testing the Software Project Management Plan	292
	Chapter Review	292
	For Further Reading	292
	Key Terms	293
	Problems	294
	References	295
PART B		
THE WORKFLOWS OF THE SOFTWARE LIFE CYCLE 299		
Chapter 10		
Key Material from Part A 301		
Learning Objective 301		
10.1	Software Development: Theory versus Practice	301
10.2	Iteration and Incrementation	302
10.3	The Unified Process	306
10.4	Workflow Overview	307
10.5	Teams	307
10.6	Cost – Benefit Analysis	308
10.7	Metrics	308
10.8	CASE	308
10.9	Versions and Configurations	309
10.10	Testing Terminology	309
10.11	Execution-Based and Non-Execution-Based Testing	309
10.12	Modularity	310
10.13	Reuse	310
10.14	Software Project Management Plan	310
	Chapter Review	311
	Key Terms	311
	Problems	312
Chapter 11		
Requirements 313		
Learning Objectives 313		
11.1	Determining What the Client Needs	313
11.2	Overview of the Requirements Workflow	314
11.3	Understanding the Domain	315
11.4	The Business Model	316
11.4.1	Interviewing	316

11.4.2 Other Techniques	317
11.4.3 Use Cases	318
11.5 Initial Requirements	319
11.6 Initial Understanding of the Domain: The MSG Foundation Case Study	320
11.7 Initial Business Model: The MSG Foundation Case Study	322
11.8 Initial Requirements: The MSG Foundation Case Study	326
11.9 Continuing the Requirements Workfl ow: The MSG Foundation Case Study	328
11.10 Revising the Requirements: The MSG Foundation Case Study	330
11.11 The Test Workfl ow: The MSG Foundation Case Study	338
11.12 The Classical Requirements Phase	347
11.13 Rapid Prototyping	348
11.14 Human Factors	349
11.15 Reusing the Rapid Prototype	351
11.16 CASE Tools for the Requirements Workfl ow	353
11.17 Metrics for the Requirements Workfl ow	353
11.18 Challenges of the Requirements Workfl ow	354
Chapter Review	355
For Further Reading	356
Key Terms	357
Case Study Key Terms	357
Problems	357
References	358
Chapter 12	
Classical Analysis	360
Learning Objectives	360
12.1 The Specifi cation Document	360
12.2 Informal Specifi cations	362
12.2.1 Correctness Proof Mini Case Study Redux	363
12.3 Structured Systems Analysis	364
12.3.1 Sally ' s Software Shop Mini Case Study	364
12.4 Structured Systems Analysis: The MSG Foundation Case Study	372
12.5 Other Semiformal Techniques	373
12.6 Entity-Relationship Modeling	374
12.7 Finite State Machines	376
12.7.1 Finite State Machines: The Elevator Problem Case Study	378
12.8 Petri Nets	382
12.8.1 Petri Nets: The Elevator Problem Case Study	385
12.9 Z	387
12.9.1 Z: The Elevator Problem Case Study	388
12.9.2 Analysis of Z	390
12.10 Other Formal Techniques	392
12.11 Comparison of Classical Analysis Techniques	392
12.12 Testing during Classical Analysis	393
12.13 CASE Tools for Classical Analysis	394
12.14 Metrics for Classical Analysis	395
12.15 Software Project Management Plan: The MSG Foundation Case Study	395
12.16 Challenges of Classical Analysis	396
Chapter Review	396
For Further Reading	397

Key Terms	398
Case Study Key Terms	398
Problems	398
References	400
Chapter 13	
Object-Oriented Analysis	404
Learning Objectives	404
13.1 The Analysis Workflow	405
13.2 Extracting the Entity Classes	406
13.3 Object-Oriented Analysis: The Elevator Problem Case Study	407
13.4 Functional Modeling: The Elevator Problem Case Study	407
13.5 Entity Class Modeling: The Elevator Problem Case Study	410
13.5.1 Noun Extraction	411
13.5.2 CRC Cards	413
13.6 Dynamic Modeling: The Elevator Problem Case Study	414
13.7 The Test Workflow: Object-Oriented Analysis	417
13.8 Extracting the Boundary and Control Classes	424
13.9 The Initial Functional Model: The MSG Foundation Case Study	425
13.10 The Initial Class Diagram: The MSG Foundation Case Study	428
13.11 The Initial Dynamic Model: The MSG Foundation Case Study	430
13.12 Revising the Entity Classes: The MSG Foundation Case Study	432
13.13 Extracting the Boundary Classes: The MSG Foundation Case Study	434
13.14 Extracting the Control Classes: The MSG Foundation Case Study	435
13.15 Use-Case Realization: The MSG Foundation Case Study	435
13.15.1 Estimate Funds Available for Week Use Case	436
13.15.2 Manage an Asset Use Case	442
13.15.3 Update Estimated Annual Operating Expenses Use Case	446
13.15.4 Produce a Report Use Case	449
13.16 Incrementing the Class Diagram: The MSG Foundation Case Study	454
13.17 The Test Workflow: The MSG Foundation Case Study	456
13.18 The Specification Document in the Unified Process	456
13.19 More on Actors and Use Cases	457
13.20 CASE Tools for the Object-Oriented Analysis Workflow	458
13.21 Metrics for the Object-Oriented Analysis Workflow	459
13.22 Challenges of the Object-Oriented Analysis Workflow	459
Chapter Review	460
For Further Reading	461
Key Terms	462
Problems	462
References	463
Chapter 14	
Design	465
Learning Objectives	465
14.1 Design and Abstraction	466
14.2 Operation-Oriented Design	466
14.3 Data Flow Analysis	467
14.3.1 Mini Case Study Word Counting	468
14.3.2 Data Flow Analysis Extensions	473
14.4 Transaction Analysis	473

14.5 Data-Oriented Design	475
14.6 Object-Oriented Design	476
14.7 Object-Oriented Design: The Elevator Problem Case Study	477
14.8 Object-Oriented Design: The MSG Foundation Case Study	481
14.9 The Design Workflow	483
14.10 The Test Workflow: Design	487
14.11 The Test Workflow: The MSG Foundation Case Study	488
14.12 Formal Techniques for Detailed Design	488
14.13 Real-Time Design Techniques	488
14.14 CASE Tools for Design	490
14.15 Metrics for Design	490
14.16 Challenges of the Design Workflow	491
Chapter Review	492
For Further Reading	493
Key Terms	493
Problems	494
References	495
Chapter 15	
Implementation	498
Learning Objectives	498
15.1 Choice of Programming Language	498
15.2 Fourth-Generation Languages	501
15.3 Good Programming Practice	504
15.3.1 Use of Consistent and Meaningful Variable Names	504
15.3.2 The Issue of Self-Documenting Code	505
15.3.3 Use of Parameters	507
15.3.4 Code Layout for Increased Readability	507
15.3.5 Nested if Statements	507
15.4 Coding Standards	509
15.5 Code Reuse	510
15.6 Integration	510
15.6.1 Top-down Integration	511
15.6.2 Bottom-up Integration	513
15.6.3 Sandwich Integration	513
15.6.4 Integration of Object-Oriented Products	514
15.6.5 Management of Integration	515
15.7 The Implementation Workflow	516
15.8 The Implementation Workflow: The MSG Foundation Case Study	516
15.9 The Test Workflow: Implementation	516
15.10 Test Case Selection	517
15.10.1 Testing to Specifications versus Testing to Code	517
15.10.2 Feasibility of Testing to Specifications	517
15.10.3 Feasibility of Testing to Code	518
15.11 Black-Box Unit-Testing Techniques	520
15.11.1 Equivalence Testing and Boundary Value Analysis	521
15.11.2 Functional Testing	522
15.12 Black-Box Test Cases: The MSG Foundation Case Study	523
15.13 Glass-Box Unit-Testing Techniques	525
15.13.1 Structural Testing: Statement, Branch, and Path Coverage	526

15.13.2 Complexity Metrics	527
15.14 Code Walkthroughs and Inspections	528
15.15 Comparison of Unit-Testing Techniques	528
15.16 Cleanroom	529
15.17 Potential Problems When Testing Objects	530
15.18 Management Aspects of Unit Testing	533
15.19 When to Reimplement Rather than Debug a Code Artifact	533
15.20 Integration Testing	535
15.21 Product Testing	535
15.22 Acceptance Testing	536
15.23 The Test Workflow: The MSG Foundation Case Study	537
15.24 CASE Tools for Implementation	537
15.24.1 CASE Tools for the Complete Software Process	538
15.24.2 Integrated Development Environments	538
15.24.3 Environments for Business Applications	539
15.24.4 Public Tool Infrastructures	540
15.24.5 Potential Problems with Environments	540
15.25 CASE Tools for the Test Workflow	540
15.26 Metrics for the Implementation Workflow	541
15.27 Challenges of the Implementation Workflow	542
Chapter Review	542
For Further Reading	543
Key Terms	544
Problems	545
References	547
Chapter 16	
Postdelivery Maintenance	551
Learning Objectives	551
16.1 Development and Maintenance	551
16.2 Why Postdelivery Maintenance Is Necessary	553
16.3 What Is Required of Postdelivery Maintenance Programmers?	553
16.4 Postdelivery Maintenance Mini Case Study	555
16.5 Management of Postdelivery Maintenance	557
16.5.1 Defect Reports	557
16.5.2 Authorizing Changes to the Product	558
16.5.3 Ensuring Maintainability	559
16.5.4 Problem of Repeated Maintenance	559
16.6 Maintenance of Object-Oriented Software	560
16.7 Postdelivery Maintenance Skills versus Development Skills	563
16.8 Reverse Engineering	563
16.9 Testing during Postdelivery Maintenance	564
16.10 CASE Tools for Postdelivery Maintenance	565
16.11 Metrics for Postdelivery Maintenance	566
16.12 Postdelivery Maintenance: The MSG Foundation Case Study	566
16.13 Challenges of Postdelivery Maintenance	566
Chapter Review	566
For Further Reading	567
Key Terms	567

Problems	567
References	568
Chapter 17	
More on UML	571
Learning Objectives	571
17.1 UML Is Not a Methodology	571
17.2 Class Diagrams	572
17.2.1 Aggregation	573
17.2.2 Multiplicity	574
17.2.3 Composition	575
17.2.4 Generalization	576
17.2.5 Association	576
17.3 Notes	577
17.4 Use-Case Diagrams	577
17.5 Stereotypes	577
17.6 Interaction Diagrams	579
17.7 Statecharts	581
17.8 Activity Diagrams	583
17.9 Packages	585
17.10 Component Diagrams	586
17.11 Deployment Diagrams	586
17.12 Review of UML Diagrams	587
17.13 UML and Iteration	587
Chapter Review	587
For Further Reading	588
Key Terms	588
Problems	588
References	589
Chapter 18	
Emerging Technologies	590
Learning Objectives	590
18.1 Aspect-Oriented Technology	591
18.2 Model-Driven Technology	593
18.3 Component-Based Technology	594
18.4 Service-Oriented Technology	594
18.5 Comparison of Service-Oriented and Component-Based Technology	595
18.6 Social Computing	596
18.7 Web Engineering	596
18.8 Cloud Technology	597
18.9 Web 3.0	598
18.10 Computer Security	598
18.11 Model Checking	598
18.12 Present and Future	599
Chapter Review	599
For Further Reading	599
Key Terms	599
References	600
Bibliography	601
Appendix A	

Term Project: Chocoholics Anonymous 627

Appendix B

Software Engineering Resources 630

Appendix C

Requirements Workflow: The MSG Foundation Case Study 632

Appendix D

Structured Systems Analysis: The MSG Foundation Case Study 633

Appendix E

Analysis Workflow: The MSG Foundation Case Study 636

Appendix F

Software Project Management Plan: The MSG Foundation Case Study 637

Appendix G

Design Workflow: The MSG Foundation Case Study 642

Appendix H

Implementation Workflow: The MSG Foundation Case Study (C++ Version) 647

Appendix I

Implementation Workflow: The MSG Foundation Case Study (Java Version) 648

Appendix J

Test Workflow: The MSG Foundation Case Study 649

Author Index 651

Subject Index 654

章节摘录

版权页：插图：A word used on almost every page of this book is software. Software consists of not just code in machine-readable form but also all the documentation that is an intrinsic component of every project. Software includes the specification document, the design document, legal and accounting documents of all kinds, the software project management plan, and other management documents as well as all types of manuals. Since the 1970s, the difference between a program and a system has become blurred. In the "good old days," the distinction was clear. A program was an autonomous piece of code, generally in the form of a deck of punched cards that could be executed. A system was a related collection of programs. A system might consist of programs P, Q, R, and S. Magnetic tape T was mounted, and then program P was run. It caused a deck of data cards to be read in and produced as output tapes T2 and Ta. Tape T2 then was rewound, and program Q was run, producing tape T4 as output. Program R now merged tapes Ta and T4 into tape Ts; Ts served as input for program S, which printed a series of reports. Compare that situation with a product, running on a machine with a front-end communications processor and a back-end database manager, that performs real-time control of a steel mill. The single piece of software controlling the steel mill does far more than the old-fashioned system, but in terms of the classic definitions of program and system, this software undoubtedly is a program. To add to the confusion, the term system now is also used to denote the hardware-software combination. For example, the flight control system in an aircraft consists of both the in-flight computers and the software running on them. Depending on who is using the term, the flight control system also may include the controls, such as the joystick, that send commands to the computer and the parts of the aircraft, such as the wing flaps, controlled by the computer. Furthermore, within the context of traditional software development, the term systems analysis refers to the first two phases (requirements and analysis phases) and systems design refers to the third phase (design phase). To minimize confusion, this book uses the term product to denote a nontrivial piece of software. There are two reasons for this convention. The first is simply to obviate the program versus system confusion by using a third term. The second reason is more important. This book deals with the process of software production, that is, the way we produce software, and the end result of a process is termed a product. Finally, the term system is used in its modern sense, that is, the combined hardware and software, or as part of universally accepted phrases, such as operating system and management information system. Two words widely used within the context of software engineering are methodology and paradigm. In the 1970s, the word methodology began to be used in the sense of "a way of developing a software product"; the word actually means the "science of methods."

精彩短评

- 1、没什么优缺点可说的！
- 2、这本书是学校要求我们买的教材，比学校的价格优惠，但是优惠力度还不够喔，希望以后亚马逊加大力度搞活动啦
- 3、应该是高仿 纸质比我同学从学校买的正版的略差

版权说明

本站所提供下载的PDF图书仅提供预览和简介，请支持正版图书。

更多资源请访问：www.tushu000.com