原版风暴 · 软件工程系列

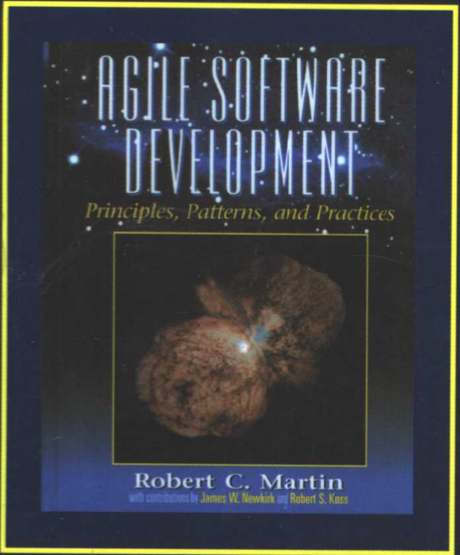
PEARSON Prentice Hall

Agile Software Development
Principles, Patterns, and Practices

敏捷软件开发
（影印版）

2003 年度 Jolt 大奖

［美］Robert C. Martin 著



软件开发和管理人员必读经典 ▪

《Refactoring》作者 Martin Fowler 全力推荐 ▪

原汁原味，零距离领悟大师思想精髓 ▪

Pearson Education

中国电力出版社
www.infopower.com.cn

原版风暴 • 软件工程系列

Agile Software Development

Principles, Patterns, and Practices

敏捷软件开发

（影印版）

[美] Robert C. Martin 著

中国电力出版社

Agile Software Development

Principles, Patterns, and Practices

Robert Cecil Martin

图书在版编目（CIP）数据
敏捷软件开发 / （美）马丁著．—影印本．—北京：中国电力出版社，2003.5
（原版风暴 • 软件工程系列）
ISBN 7-5083-1503-0
Ⅰ.敏… Ⅱ.马… Ⅲ.软件开发－英文 Ⅳ.TP311.52
中国版本图书馆 CIP 数据核字（2003）第 027801 号
出版者：中国电力出版社（北京西城区三里河路 6 号，邮编 100044）
http://www.infopower.com.cn

责任编辑：陈维宁
丛 书 名：原版风暴 • 软件工程系列
书　　名：敏捷软件开发（影印版）
编　　著：（美）Robert C. Martin
出 版 者：中国电力出版社
地址：北京市三里河路6号　邮政编码：100044
电话：（010）88515918　传真：（010）88423191
印　　刷：汇鑫印务有限公司印刷
发 行 者：新华书店总店北京发行所
开　　本：787×1092 1/16　　**印　　张：**35
书　　号：ISBN 7-5083-1503-0
版　　次：2003年7月北京第一版
印　　次：2003年7月第一次印刷
定　　价：59.00 元

Foreword

I'm writing this foreword right after having shipped a major release of the Eclipse open source project. I'm still in recovery mode, and my mind is bleary. But one thing remains clearer than ever: that people, not processes, are the key to shipping a product. Our recipe for success is simple: work with individuals obsessed with shipping software, develop with lightweight processes that are tuned to each team, and adapt constantly.

Double-clicking on developers from our teams reveals individuals who consider programming the focus of development. Not only do they write code; they digest it constantly to maintain an understanding of the system. Validating designs with code provides feedback that's crucial for getting confidence in a design. At the same time, our developers understand the importance of patterns, refactoring, testing, incremental delivery, frequent builds, and other best-practices of XP that have altered the way we view methodologies today.

Skill in this style of development is a prerequisite for success in projects with high technical risk and changing requirements. Agile development is low-key on ceremony and project documentation, but it's intense when it comes to the day-to-day development practices that count. Putting these practices to work is the focus of this book.

Robert is a longtime activist in the object-oriented community, with contributions to C++ practice, design patterns, and object-oriented design principles in general. He was an early and vocal advocate of XP and agile methods. This book builds on these contributions, covering the full spectrum of agile development practice. It's an ambitious effort. Robert makes it more so by demonstrating everything through case studies and lots of code, as befits agile practice. He explains programming and design by actually doing it.

This book is crammed with sensible advice for software development. It's equally good whether you want to become an agile developer or improve the skills you already have. I was looking forward to this book, and I wasn't disappointed.

Erich Gamma
Object Technology International

For Ann Marie, Angela, Micah, Gina, Justin, Angelique, Matt, and Alexis . . .

There is no greater treasure,
Nor any wealthier trove,
Than the company of my family,
And the comfort of their love.

Preface

But Bob, you said you'd be done with the book last year.

—Claudia Frers, *UML World*, 1999

Agile development is the ability to develop software quickly, in the face of rapidly changing requirements. In order to achieve this agility, we need to employ practices that provide the necessary discipline and feedback. We need to employ design principles that keep our software flexible and maintainable, and we need to know the design patterns that have been shown to balance those principles for specific problems. This book is an attempt to knit all three of these concepts together into a functioning whole.

This book describes those principles, patterns, and practices and then demonstrates, how they are applied by walking through dozens of different case studies. More importantly, the case studies are not presented as complete works. Rather, they are designs *in progress*. You will see the designers make mistakes, and you will observe how they identify the mistakes and eventually correct them. You will see them puzzle over conundrums and worry over ambiguities and trade-offs. You will see the *act* of design.

The Devil Is in the Details

This book contains a *lot* of Java and C++ code. I hope you will carefully read that code since, to a large degree, the code is the *point* of the book. The code is the actualization of what this book has to say.

There is a repeating pattern to this book. It consists of a series of case studies of varying sizes. Some are very small, and some require several chapters to describe. Each case study is preceded by material that is meant to prepare you for it. For example, the Payroll case study is preceded by chapters describing the object-oriented design principles and patterns used in the case study.

The book begins with a discussion of development practices and processes. That discussion is punctuated by a number of small case studies and examples. From there, the book moves on to the topic of design and design principles, and then to some design patterns, more design principles that govern packages, and more patterns. All of these topics are accompanied by case studies.

So prepare yourself to read some code and to pore over some UML diagrams. The book you are about to read is *very* technical, and its lessons, like the devil, are in the details.

A Little History

Over six years ago, I wrote a book entitled *Designing Object-Oriented C++ Applications using the Booch Method.* It was something of magnum opus for me, and I was very pleased with the result and with the sales.

This book started out as a second edition to *Designing*, but that's not how it turned out. Very little remains of the original book in these pages. Little more than three chapters have been carried through, and those chapters have been massively changed. The intent, spirit, and many of the lessons of the book are the same. And yet, I've learned a tremendous amount about software design and development in the six years since *Designing* came out. This book reflects that learning.

What a half-decade! *Designing* came out just before the Internet collided with the planet. Since then, the number of abbreviations we have to deal with has doubled. We have Design Patterns, Java, EJB, RMI, J2EE, XML, XSLT, HTML, ASP, JSP, Servlets, Application Servers, ZOPE, SOAP, C#, .NET, etc., etc. Let me tell you, it's been hard to keep the chapters of this book reasonably current!

The Booch Connection

In 1997, I was approached by Grady Booch to help write the third edition of his amazingly successful *Object-Oriented Analysis and Design with Applications.* I had worked with Grady before on some projects, and I had been an avid reader and contributor to his various works, including UML. So I accepted with glee. I asked my good friend Jim Newkirk to help out with the project.

Over the next two years, Jim and I wrote a number of chapters for the Booch book. Of course, that effort meant that I could not put as much effort into this book as I would have liked, but I felt that the Booch book was worth the contribution. Besides, this book was really just a second edition of *Designing* at the time, and my heart wasn't in it. If I was going to say something, I wanted to say something new and different.

Unfortunately, that version of the Booch book was not to be. It is hard to find the time to write a book during normal times. During the heady days of the ".com" bubble, it was nearly impossible. Grady got ever busier with Rational and with new ventures like Catapulse. So the project stalled. Eventually, I asked Grady and Addison-Wesley if I could have the chapters that Jim and I wrote to include in *this* book. They graciously agreed. So several of the case study and UML chapters came from that source.

The Impact of Extreme Programming

In late 1998, XP reared its head and challenged our cherished beliefs about software development. Should we create lots of UML diagrams prior to writing any code, or should we eschew any kind of diagrams and just write lots of code? Should we write lots of narrative documents that describe our design, or should we try to make the *code* narrative and expressive so that ancillary documents aren't necessary? Should we program in pairs? Should we write tests before we write production code? What should we do?

This revolution came at an opportune time for me. During the middle to late 90s, Object Mentor was helping quite a few companies with object-oriented (OO) design and project management issues. We were helping companies get their projects *done*. As part of that help, we instilled our own attitudes and practices into the teams. Unfortunately, these attitudes and practices were not written down. Rather, they were an oral tradition that was passed from us to our customers.

By 1998, I realized that we needed to write down our process and practices so that we could better articulate them to our customers. So, I wrote many articles about process in the C++ *Report.*[1] These articles missed the mark. They were informative, and in some cases entertaining, but instead of codifying the practices and attitudes

1. These articles are available in the "publications" section of `http://www.objectmentor.com`. There are four of them. The first three are entitled "Iterative and Incremental Development" (I, II, III). The last is entitled "C.O.D.E Culled Object Development procEss."

that we actually used in our projects, they were an unwitting compromise to values that had been imposed upon me for decades. It took Kent Beck to show me that.

The Beck Connection

In late 1998, as I was fretting over codifying the Object-Mentor process, I ran into Kent's work on Extreme Programming (XP). The work was scattered through Ward Cunningham's *wiki*[2] and was mixed with the writings of many others. Still, with some work and diligence I was able to get the gist of what Kent was talking about. I was intrigued, but skeptical. Some of the things that XP talked about were exactly on target for my concept of a development process. Other things, however, like the lack of an articulated design step, left me puzzled.

Kent and I could not have come from more disparate software circumstances. He was a recognized Smalltalk consultant, and I was a recognized C++ consultant. Those two worlds found it difficult to communicate with one another. There was an almost Kuhnian[3] paradigm gulf between them.

Under other circumstances, I would never have asked Kent to write an article for the *C++ Report*. But the congruence of our thinking about process was able to breech the language gulf. In February of 1999, I met Kent in Munich at the OOP conference. He was giving a talk on XP in the room across from where I was giving a talk on principles of OOD. Being unable to hear that talk, I sought Kent out at lunch. We talked about XP, and I asked him to write an article for the *C++ Report*. It was a great article about an incident in which Kent and a coworker had been able to make a sweeping design change in a live system in a matter of an hour or so.

Over the next several months, I went through the slow process of sorting out my own fears about XP. My greatest fear was in adopting a process in which there is no explicit up-front design step. I found myself balking at that. Didn't I have an obligation to my clients, and to the industry as a whole, to teach them that design is important enough to spend time on?

Eventually, I realized that I did not really practice such a step myself. Even in all the articles and books I had written about design, Booch diagrams, and UML diagrams, I had always used code as a way to verify that the diagrams were meaningful. In all my customer consulting, I would spend an hour or two helping them to draw diagrams and then I would direct them to explore those diagrams with code. I came to understand that though XP's words about design were foreign (in a Kuhnian[4] sense), the practices behind the words were familiar to me.

My other fears about XP were easier to deal with. I had always been a closet pair programmer. XP gave me a way to come out of the closet and revel in my desire to program with a partner. Refactoring, continuous integration, and customer on-site were all very easy for me to accept. They were very close to the way I already advised my customers to work.

One practice of XP was a revelation for me. Test-first design sounds innocuous when you first hear it. It says to write test cases before you write production code. All production code is written to make failing test cases pass. I was not prepared for the profound ramifications that writing code this way would have. This practice has completely transformed the way I write software, and transformed it for the better. You can see that transformation in this book. Some of the code written in this book was written before 1999. You won't find test cases for that code. On the other hand, all of the code written after 1999 is presented with test cases, and the test cases are typically presented first. I'm sure you'll note the difference.

So, by the fall of 1999 I was convinced that Object Mentor should adopt XP as its process of choice and that I should let go of my desire to write my own process. Kent had done an excellent job of articulating the practices and process of XP, and my own feeble attempts paled in comparison.

2. `http://c2.com/cgi/wiki`. This website contains a vast number of articles on an immense variety of subjects. Its authors number in the hundreds or thousands. It has been said that only Ward Cunningham could instigate a social revolution using a few lines of Perl.

3. Any credible intellectual work written between 1995 and 2001 must use the term "Kuhnian." It refers to the book, *The Structure of Scientific Revolutions*, by Thomas S. Kuhn, The University of Chicago Press, 1962.

4. If you mention Kuhn twice in a paper, you get extra credit.

Organization

This book is organized into six major sections followed by several appendices.

- Section 1: *Agile Development.*
  This section describes the concept of agile development. It starts with the Manifesto of the Agile Alliance, provides an overview of Extreme Programming (XP), and then goes into many small case studies that illuminate some of the individual XP practices—especially those that have an impact upon the way we design and write code.
- Section 2: *Agile Design*
  The chapters in this section talk about object-oriented software design. The first chapter asks the question, *What is Design?* It discusses the problem of, and techniques for, managing complexity. Finally, the section culminates with the *principles of object-oriented class design.*
- Section 3: *The Payroll Case Study*
  This is the largest and most complete case study in the book. It describes the object-oriented design and C++ implementation of a simple batch payroll system. The first few chapters in this section describe the design patterns that the case study encounters. The final two chapters contain the full case study.
- Section 4: *Packaging the Payroll System*
  This section begins by describing the *principles of object-oriented package design.* It then goes on to illustrate those principles by incrementally packaging the classes from the previous section.
- Section 5: *The Weather Station Case Study*
  This section contains one of the case studies that was originally planned for the Booch book. The Weather Station study describes a company that has made a significant business decision and explains how the Java development team responds to it. As usual, the section begins with a description of the design patterns that will be used and then culminates in the description of the design and implementation.
- Section 6: *The ETS Case Study*
  This section contains a description of an actual project that the author participated in. This project has been in production since 1999. It is the automated test system used to deliver and score the registry examination for the National Council of Architectural Registration Boards.
- UML Notation Appendices
  The first two appendices contains several small case studies that are used to describe the UML notation.
- Miscellaneous Appendices

How to Use This Book

If You are a Developer...

Read the book cover to cover. This book was written primarily for developers, and it contains the information you need to develop software in an agile manner. Reading the book cover to cover introduces practices, then principles, then patterns, and then it provides case studies that tie them all together. Integrating all this knowledge will help you get your projects *done.*

If You Are a Manager or Business Analyst...

Read Section 1, *Agile Development.* The chapters in this section provide an in-depth discussion of agile principles and practices. They'll take you from requirements to planning to testing, refactoring, and programming. It will give you guidance on how to build teams and manage projects. It will help you get your projects *done.*

If You Want to Learn UML...

First read Appendix A, *UML Notation I: The CGI Example.* Then read Appendix B, *UML Notation II: The STATMUX.* Then, read all the chapters in Section 3, *The Payroll Case Study.* This course of reading will give you a good grounding in both the syntax and use of UML. It will also help you translate between UML and a programming language like Java or C++.

If You Want to Learn Design Patterns...

To find a particular pattern, use the "List of Design Patterns" on page xxii to find the pattern you are interested in.

To learn about patterns in general, read Section 2, *Agile Design* to first learn about design principles, and then read Section 3, *The Payroll Case Study*; Section 4, *Packaging the Payroll System*; Section 5, *The Weather Station Case Study*; and Section 6, *The ETS Case Study.* These sections define all the patterns and show how to use them in typical situations.

If You Want to Learn about Object-Oriented Design Principles...

Read Section 2, *Agile Design*; Section 3, *The Payroll Case Study*; and Section 4, *Packaging the Payroll System.* These chapters will describe the principles of object-oriented design and will show you how to use them.

If You Want to Learn about Agile Development Methods...

Read Section 1, *Agile Development.* This section describes agile development from requirements to planning, testing, refactoring, and programming.

If You Want a Chuckle or Two...

Read Appendix C, *A Satire of Two Companies.*

Acknowledgments

A heartfelt thanks to:

Lowell Lindstrom, Brian Button, Erik Meade, Mike Hill, Michael Feathers, Jim Newkirk, Micah Martin, Angelique Thouvenin Martin, Susan Rosso, Talisha Jefferson, Ron Jeffries, Kent Beck, Jeff Langr, David Farber, Bob Koss, James Grenning, Lance Welter, Pascal Roy, Martin Fowler, John Goodsen, Alan Apt, Paul Hodgetts, Phil Markgraf, Pete McBreen, H. S. Lahman, Dave Harris, James Kanze, Mark Webster, Chris Biegay, Alan Francis, Fran Daniele, Patrick Lindner, Jake Warde, Amy Todd, Laura Steele, William Pietr, Camille Trentacoste, Vince O'Brien, Gregory Dulles, Lynda Castillo, Craig Larman, Tim Ottinger, Chris Lopez, Phil Goodwin, Charles Toland, Robert Evans, John Roth, Debbie Utley, John Brewer, Russ Ruter, David Vydra, Ian Smith, Eric Evans, everyone in the Silicon Valley Patterns group, Pete Brittingham, Graham Perkins, Phlip, and Richard MacDonald.

The books reviewers:

Pete McBreen / McBreen Consulting	Bjarne Stroustrup / AT & T Research
Stephen J. Mellor / Projtech.com	Micah Martin / Object Mentor Inc.
Brian Button / Object Mentor Inc.	James Grenning / Object Mentor Inc.

A very special thanks to Grady Booch and Paul Becker for allowing me to include chapters that were originally slated for Grady's third edition of *Object Oriented Analysis and Design with Applications.*

A special thanks to Jack Reeves for graciously allowing me to reproduce his "What is Design?" article.

Another special thanks to Erich Gamma, for writing the foreword to this book. I hope the fonts are better this time Erich!

The wonderful and sometimes dazzling illustrations at the head of each chapter were drawn by Jennifer Kohnke. The decorative illustrations scattered throughout the midst of the chapters are the lovely product of Angela Dawn Martin Brooks, my daughter, and one of the joys of my life.

Resources

All the source code in this book can be downloaded from `www.objectmentor.com/PPP`.

About the Authors

Robert C. Martin

Robert C. Martin (Uncle Bob) has been a software professional since 1970 and an international software consultant since 1990. He is founder and president of Object Mentor Inc., a team of experienced consultants who mentor their clients worldwide in the fields of C++, Java, .NET, OO, Patterns, UML, Agile Methodologies, and Extreme Programming. In 1995, Robert authored the best-selling book: *Designing Object Oriented C++ Applications using the Booch Method*, published by Prentice Hall. From 1996 to 1999 he was the editor-in-chief of the *C++ Report*. In 1997, he was chief editor of the book: *Pattern Languages of Program Design 3*, published by Addison–Wesley. In 1999, he was the editor of *More C++ Gems* published by Cambridge Press. He is co-author, with James Newkirk, of *XP in Practice*, Addision–Wesley, 2001. In 2002, he wrote the long awaited *Agile Software Development: Principles, Patterns, and Practices*, Prentice Hall, 2002. He has published dozens of articles in various trade journals, and is a regular speaker at international conferences and trade shows. And he's as happy as a clam.

James W. Newkirk

James Newkirk is a Software Development Manager/Architect. His eighteen years of experience ranges from programming real-time micro-controllers to web services. He co-wrote *Extreme Programming in Practice*, published by Addison–Wesley, 2001. Since August of 2000 he has been working with the .NET Framework and has contributed to the development of NUnit, a unit-testing tool for .NET.

Robert S. Koss

Robert S. Koss, Ph.D., has been writing software for 29 years. He has applied the principles of Object Oriented Design to many projects where he has served in roles ranging from programmer to senior architect. Dr. Koss has taught hundreds of OOD and programming language courses to thousands of students throughout the world. He is currently employed as a Senior Consultant at Object Mentor, Inc.

Brief Contents

Contents