HZ BOOKS

PEARSON Addison Wesley

经　典　原　版　书　库

软件需求管理 用例方法

（英文版 · 第2版）

MANAGING SOFTWARE REQUIREMENTS

SECOND EDITION

A USE CASE APPROACH

DEAN LEFFINGWELL
DON WIDRIG

Foreword by Ed Yourdon

OBJECT TECHNOLOGY SERIES
BOOCH
JACOBSON
RUMBAUGH
ADDISON-WESLEY
SERIES EDITORS

（美） Dean Leffingwell
Don Widrig 著

机械工业出版社
China Machine Press

经 典 原 版 书 库

软件需求管理
用例方法
（英文版·第2版）

Managing Software Requirements
A Use Case Approach
(Second Edition)

（美） Dean Leffingwell
Don Widrig 著

机械工业出版社
China Machine Press

本书版权登记号：图字：01-2004-0312

图书在版编目（CIP）数据

软件需求管理：用例方法（英文版・第2版）/（美）莱芬韦尔（Leffingwell, D.）等著. -北京：机械工业出版社，2004.2

（经典原版书库）

书名原文：Managing Software Requirements: A Use Case Approach, Second Edition

ISBN 7-111-13787-6

Ⅰ.软… Ⅱ.莱… Ⅲ.软件开发-英文 Ⅳ.TP311.52

中国版本图书馆CIP数据核字（2003）第126648号

机械工业出版社（北京市西城区百万庄大街22号 邮政编码 100037）

责任编辑：迟振春

北京昌平奔腾印刷厂印刷・新华书店北京发行所发行

2004年2月第1版第1次印刷

787mm × 1092mm 1/16・33.75印张

印数：0 001 - 3 000册

定价：55.00元

出版者的话

文艺复兴以降，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的传统，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还揭橥了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短、从业人员较少的现状下，美国等发达国家在其计算机科学发展的几十年间积淀的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章图文信息有限公司较早意识到“出版要为教育服务”。自1998年开始，华章公司就将工作重点放在了遴选、移译国外优秀教材上。经过几年的不懈努力，我们与Prentice Hall，Addison-Wesley，McGraw-Hill，Morgan Kaufmann等世界著名出版公司建立了良好的合作关系，从它们现有的数百种教材中甄选出Tanenbaum，Stroustrup，Kernighan，Jim Gray等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及庋藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力襄助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专诚为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍，为进一步推广与发展打下了坚实的基础。

随着学科建设的初步完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都步入一个新的阶段。为此，华章公司将加大引进教材的力度，在“华章教育”的总规划之下出版三个系列的计算机教材：除“计算机科学丛书”之外，对影印版的教材，则单独开辟出“经典原版书库”；同时，引进全美通行的教学辅导书“Schaum's Outlines”系列组成“全美经典学习指导系列”。为了保证这三套丛书的权威性，同时也为了更好地为学校和老师们服务，华章公司聘请了中国科学院、北京大学、清华大学、国防科技大学、复旦大学、上海交通大学、南京大学、浙江大学、中国科技大学、哈尔滨工业大学、西安交通大学、中国人民大学、北京航空航天大学、北京邮电大学、中山大学、解放军理工大学、郑州大学、湖北工学院、中国国

家信息安全测评认证中心等国内重点大学和科研机构在计算机的各个领域的著名学者组成“专家指导委员会”，为我们提供选题意见和出版监督。

这三套丛书是响应教育部提出的使用外版教材的号召，为国内高校的计算机及相关专业的教学度身订造的。其中许多教材均已为M. I. T.，Stanford，U.C. Berkeley，C. M. U. 等世界名牌大学所采用。不仅涵盖了程序设计、数据结构、操作系统、计算机体系结构、数据库、编译原理、软件工程、图形学、通信与网络、离散数学等国内大学计算机专业普遍开设的核心课程，而且各具特色——有的出自语言设计者之手、有的历经三十年而不衰、有的已被全世界的几百所高校采用。在这些圆熟通博的名师大作的指引之下，读者必将在计算机科学的宫殿中由登堂而入室。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证，但我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。教材的出版只是我们的后续服务的起点。华章公司欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下：

电子邮件：hzedu@hzbook.com
联系电话：(010) 68995264
联系地址：北京市西城区百万庄南街1号
邮政编码：100037

This book is dedicated to Becky, my wife and my best friend.

—Dean

This book is dedicated to my book partner, Dean, who made this effort worthwhile,
and to my life partner, Barbara, who makes life worthwhile.

—Don

FOREWORD

By Ed Yourdon

THE ROCK PROBLEM

One of my students summarized the issues discussed in this book as the "rock" problem. She works as a software engineer in a research laboratory, and her customers often give her project assignments that she describes as "Bring me a rock." But when you deliver the rock, the customer looks at it for a moment and says, "Yes, *but*, actually, what I really wanted was a *small blue* rock." The delivery of a small blue rock elicits the further request for a *spherical* small blue rock.

Ultimately, it may turn out that the customer was thinking all along of a small blue marble. Or maybe he wasn't sure what he wanted, but a small blue marble would have sufficed.

At each subsequent meeting with the customer, the developer may exclaim, "You want it to do *what?*" The developer is frustrated because she had something entirely different in mind when she worked long and hard to produce a rock with the characteristics she thought the customer said he needed; the customer is equally frustrated because he's convinced that he has expressed it clearly. These developers just don't get it!

To complicate matters, in most real projects, more than two individuals are involved. In addition to the customer and the developer—who may, of course, have very different names and titles—there are likely to be marketing people, testing and quality assurance people, product managers, general managers, and a variety of "stakeholders" whose day-to-day operations will be affected by the development of the new system.

All of these people can become frustrated by the problems of specifying an acceptable "rock," particularly because there often isn't enough time in today's competitive, fast-moving business world to scrap an expensive, two-year "rock project" and do it all over again. We've got to get it right the first time yet also provide for the iterative process in which the customer ultimately discovers what kind of rock he wants.

It's difficult enough to do this when we're dealing with tangible, physical artifacts like rocks. Most business organizations and government agencies today are "information intensive," so even if they're nominally in the business of building and selling rocks, there's a good chance that the rock contains an embedded computer system. Even if it doesn't, there's a good chance that the business needs elaborate systems to keep track of its e-commerce rock sales, its rock customers, its rock competitors and suppliers, and all of the other information that it needs to remain competitive in the rock business. Moreover, for thousands of companies today, those companies whose business is dedicated exclusively to the development and sales of *software products*, their entire business focuses on making their products—intangible and abstract as they are—into tangible rocks that their customers can purchase, evaluate, and apply.

Software systems, by their nature, are intangible, abstract, complex, and—in theory, at least—"soft" and infinitely changeable. So, if the customer begins articulating vague requirements for a "rock system," he often does this on the assumption that he can clarify, change, and fill in the details as time goes on. It would be wonderful if the developers—and everyone else involved in the creation, testing, deployment, and maintenance of the rock system—could accomplish this in zero time, and at zero cost, but it doesn't work that way.

In fact, it often doesn't work at all: More than half of the software systems projects taking place today are substantially over budget and behind schedule, and as much as 25 percent to 33 percent of the projects are canceled before completion, often at a staggering cost.

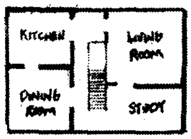
Preventing these failures and providing a rational approach for building the system the customer does want is the objective of this book. However, this is *not* a book about programming, and it's not written just for the software developer. This is a book about managing requirements for complex software applications. *As such, this book is written for every member of the software team (analysts, developers, tester and QA personnel, project management, product management, documentation folks, and the like) as well as members of the external "customer" team (users and other stakeholders, marketing, and management)*—everyone, really, who has a stake in the definition and delivery of the software system.

You'll discover that it is crucial that the members of both teams, including the nontechnical members of the external team, master the skills required to successfully define and manage the requirements process for your new system—for the simple reason that *they* are the ones who create the requirements in the first place and who ultimately determine the success or failure of the system. The stand-alone, hero programmer is an anachronism of the past: May he rest in peace.

A Simple Metaphor: Building a House

If you were a building contractor, you wouldn't need to be convinced that a series of critical conversations with the homeowner are necessary; otherwise, you might end up building a two-bedroom house when your customer wanted a three-bedroom house. But it's equally important that these "requirements" be discussed and negotiated with the government authorities concerned with building codes and zoning regulations, and you may need to check with the next-door neighbors before you decide to cut down any trees on the property where the house will be built.

The building inspector and the next-door neighbors are among the other *stakeholders* who, along with the person who intends to pay for and inhabit the house, will determine whether the finished house meets their needs. It's also clear that these important stakeholders of your system, such as neighbors and zoning officials, are not users (homeowners), and it seems equally obvious that their perspectives on what makes a quality home may differ from the homeowner's opinion.



Again, we're discussing software applications in this book, not houses or rocks. The requirements of a house might be described, at least in part, with a set of blueprints and a list of specifications; similarly, a software system can be described with models and diagrams. But just as the blueprints for a house are intended as a communication and negotiation mechanism between laypeople and engineers—and lawyers and inspectors and nosy neighbors—so the technical diagrams associated with a software system can also be created in such a way that "ordinary" people can understand them.

Many of the crucially important requirements don't need any diagrams at all. The prospective house buyer, for example, can write a requirement in ordinary English that says, "My house must have three bedrooms, and it must have a garage large enough to hold two cars and six bicycles." As you'll see in this book, the majority of the crucial requirements for a software system can be written in plain English. In other cases, it would be more helpful to have a picture of what kind of fireplace the homeowner had in mind.

Many of the *team skills* you will need to master in order to address this challenge can also be described in terms of practical, commonsense advice. "Make sure you talk to the building inspector," we might advise our novice house builder, "*before* you dig the foundation for the house, not after you've poured the cement and begun building the walls and the roof." For a software project, we would offer similar advice: "Make sure you ask the right questions of the right people, make sure that you understand how the system is going to be used, and *don't* assume that 100 percent of the requirements are critical, because you're not likely to have time to finish them all before the deadline."

About This Book

In this book, Leffingwell and Widrig have taken a pragmatic approach to describing the solution to the rock problem. They have organized the book into eight parts. The Introduction provides some of the context, definitions, and background that you'll need to understand what follows. Chapter 1 reviews the systems development "challenge." The data shows that some software project failures are indeed caused by sloppy programming, but a number of studies demonstrate that poor requirements management may be the single largest cause of project failure. And though I've described the basic concept of requirements management in a loose, informal fashion in this foreword, the authors will define it more carefully in Chapter 2, in order to lay the groundwork for the chapters that follow. Chapter 3 provides an overview of some of the software development models in use today and concludes with a recommendation for an iterative process, one that facilitates additional requirements discovery along the way. Chapter 4 provides a brief introduction to some of the characteristics of modern software teams so they can relate the team skills that will be developed to the team context, wherein the skills must be applied.

The book is structured on the six requisite team skills for effective requirements management.

Each of the next six major parts is intended to help you and your team understand and master one of the *six requisite team skills for effective requirements management.*

- To begin, of course, you will need a proper understanding of the problem that's intended to be solved with a new software system. That is addressed in Team Skill 1, Analyzing the Problem.
- Team Skill 2, Understanding User and Stakeholder Needs, is also crucial.

- Team Skill 3, Defining the System, describes the initial process of defining a system to address those requirements.
- Team Skill 4, Managing Scope, covers that absolutely crucial and often ignored process of managing the customer's expectations and the scope of the project.
- Team Skill 5, Refining the System Definition, illustrates key techniques that you will use in order to elaborate on the system to a level of detail sufficient to drive design and implementation, so the entire extended team knows exactly what kind of system you are building.
- Team Skill 6, Building the Right System, discusses the processes associated with building a system that fulfills the requirements. Team Skill 6 also discusses techniques you can use to validate that the system meets the requirements and, further, to help ensure that the system doesn't do anything malevolent to its users or otherwise exhibit unpleasant behaviors that are not defined by the requirements. And, since requirements for any nontrivial application cannot be frozen in time, the authors describe ways in which the team can actively manage change without destroying the system under construction. Team Skill 6 concludes with a chapter that suggests ways in which the requirements gathering process can improve the quality of the overall project. Special emphasis is given to the iterative nature of modern program development processes and how this yields substantial opportunities for an ongoing quality assessment.

After these descriptions of specific requirements management techniques, the authors briefly review the evolving methods of Extreme Programming and Agile Methods and demonstrate ways of integrating effective requirements management practices into the framework of these software development methods. Finally, in Chapter 31 the authors provide a prescription that you and your team can use to manage requirements in your next project.

I hope that, armed with these newly acquired team skills, you too will be able to build the perfect rock or marble. However, it will never be easy; even with the best techniques and processes, and even with automated tool support for all of this, you'll still find that it's hard work. Moreover, it's still risky; even with these team skills, some projects will fail because we're "pushing the envelope" in many organizations, attempting to build ever more complex systems in ever less time. Nevertheless, the skills defined in this book will go a long way toward reducing the risk and thereby helping you achieve the success you deserve.

Preface to the Second Edition

By Dean Leffingwell

Much has transpired since the first edition of this text was published in 1999. The "dot.com" bubble economy of the late 1990s (driven in part by the Internet, software, and related technology) has burst, causing significant disruption, economic uncertainty, and chaos in the lives of many. And yet, perhaps order and sanity have been restored to a free market that appeared to have "lost its wits" for a time.

However, innovation in software technology continues unabated, and the industry as a whole is still growing rapidly. The global reach of the Internet continues to change our lives and drive new, varied forms of communication, from the global electronic marketplaces that facilitate the exchange of goods and services to the after-school instant messaging chat-fests that seem to absorb our children's homework time and so much of that expensive Internet bandwidth we rolled out in the last decade.

We are connected to our business associates, friends, and family 24/7. Internet cafes in Australia, in Scotland, and on Alaska-bound cruise ships are open 24 hours a day. We receive e-mails on our PDAs at the grocery store. We can't make breakfast, drive to work, ride an elevator, or enter an office building without interacting with software. Software has become the embodiment of much of the world's intellectual knowledge, and the business of developing and deploying software has emerged as one of the world's most important industries.

Software development practices continue to march forward as well. The Unified Modeling Language (UML), adopted as late as 1997, is now the de facto means to communicate architecture, patterns, and design mechanisms. The Rational Unified Process and similar processes based on the UML are being adopted by many in the industry as the standard way to approach the challenge of software development.

Our personal lives have changed also. After four years at Rational Software, recently acquired by IBM, I have moved on to helping independent software companies achieve their goals. Some teams hope to change the world; some hope to have a significant impact on individual lives by improving health care; still others hope to improve their customers' manufacturing efficiencies or to help businesses grow by translating product data into other languages. However, these teams all have one thing in common: they are challenged by the difficulty of defining software solutions in a way that can be understood by themselves, by their customers, by their marketing teams, by their internal development and testing teams—indeed, by all those who must understand the proposed solution at the right level of detail so that the proper results can be achieved. Fail to do that and they fail to achieve their mission. Because of the importance of their mission on their personal lives as well as those whose products they are intended to help, failure is not an option.

Therefore, while much has changed in the software industry in just a few short years, some things, including the challenge of *Managing Software Requirements*, remain largely the same, and so our work continues in this, the second edition.

ABOUT THE SECOND EDITION

The motivation for the content changes in the second edition is based on different yet convergent factors.

The first set of factors is based on the success of the book in the marketplace, which has generated many positive comments and much encouragement, as well as constructive criticisms. While comments range widely, two consistent themes emerged.

- **The "more use cases" theme.** The first edition (subtitled *A Unified Approach*) reconciled and combined two major viewpoints on requirements techniques. The first, perhaps a more traditional approach, described the way in which requirements specifications are created and detailed to prescribe system behavior using declarative techniques ("the system shall . . ."). The second, the use case approach, described the way in which use cases could be used to define the majority of the functional behavior of the system. We combined these techniques in the first edition in order to create a common, and hopefully more holistic, approach. Based on feedback, we did achieve some success. However, one criticism of the work is that, while we recommended and described the use case method, we did not go far enough in helping the reader develop or apply this technique. Moreover, in presenting both techniques, we confused some readers who wanted to better understand which technique to apply and when.

- **The "it's a big book with many techniques—please be more prescriptive" theme.** The first edition of this book was intended to be a comprehensive work, a one-stop-shopping reference for any technique readers might need to define requirements for a system of any type. We hope this provided value to our readers because we truly believe that there is no "one size fits all" solution to each specific software engineering challenge. And yet, the reviewers' theme remains: "Does it have to be this hard? Can't you be more prescriptive?"

A second set of factors driving this same theme is based on my own experiences in using the book as I work with companies to help them achieve their software development objectives. Some have software applications that require multiple techniques; some can make time for a fairly rigorous introduction to a full requirements management discipline. However, others need to document a specific set of requirements for a specific software application and they need to do so *immediately*. Starting tomorrow. There is no time or interest in a debate about which technique might be more effective or about the nuances of anything. "Just give me one technique, make it simple, and get me started right now," they say.

Fortunately, these two sets of inputs are mostly convergent and the answer to both is fairly clear. **For most teams, in most circumstances, a combination of (1) a well-considered Vision document, (2) an identification and elaboration of the key use cases to be implemented, and (3) a supplementary specification of the nonfunctional requirements is adequate and appropriate for managing software requirements.** In addition, if this is the chosen method, the elaborated use cases can directly become the foundation for system testing.

To this end, this second edition of *Managing Software Requirements* has new content, a new theme, and a new subtitle: *A Use Case Approach*. In this edition, the use case technique is the cornerstone technique, and a more prescriptive approach has been chosen and represented. For example, Chapter 14, A Use Case Primer, has been added to provide a more fundamental basis for understanding and applying use cases. It should serve as a tutorial adequate for an otherwise uninitiated individual to be able to learn and begin to apply the technique. The HOLIS case study has also been updated to reflect a more use-case-centered approach. Chapter 26, From Use Case to Test Case, has been added to illustrate how the use cases can directly drive a comprehensive test strategy as well as serve as direct input to the test cases themselves.

In addition, we've made one substantial enhancement motivated solely by our own purposes. Chapter 17 (which appeared in the first edition as Chapter 18, The Champion), has been renamed Product Management and enhanced with new material designed to help teams understand how to turn a software application into what we call *the whole product solution.* Since getting the requirements "right" cannot by itself ensure commercial success, this chapter provides insight and guidelines for those activities (such as pricing and licensing, positioning and messaging) and other commercial factors that transform a working software application into a *software product people want to buy.*

Also, since modern software development processes are becoming more iterative, we decided to repurpose the first edition's chapter on quality so that this edition's chapter would provide a more comprehensive look at quality within the context of a modern software process. Thus Chapter 29, Assessing Requirements Quality in Iterative Development, speaks directly to iterative techniques for gathering and improving requirements within an overall iterative development framework.

Finally, we also took the opportunity to address a new undercurrent in the industry, a movement toward what are perceived as lighter, less formal methods. In the extreme, *Extreme Programming* (XP), as espoused by Beck and others, could be interpreted to eliminate process entirely. Perhaps more correctly, XP incorporates certain keystone processes, such as direct customer requirements input, directly into programming practices, but it's also fair to note that the concepts of "software process" and the "M" word (methodology) are studiously avoided. Perhaps less extreme and considered by some to be more practical, the introduction of *Agile Methods,* as advocated by Cockburn and others, has also taken root. Though controversial in some circles, these lighter approaches cannot be ignored, and we've addressed these in the requirements context in another new chapter, Chapter 30, Agile Requirements Methods.

Of course, no book can be all things to all people. In order to make this edition as readable as possible, we eliminated a number of topics and chapters from the prior version and shortened others.

We sincerely hope that you will find this revised text more approachable, as well as easier to use and apply, and that it will better help you and your teams to manage your software requirements.

ACKNOWLEDGMENTS

The authors would like to acknowledge and thank John Altinbay, Jim Heumann, and Dan Rawsthorne for their careful and insightful reviews of this second edition. We'd also to thank the many others who contributed to this work, including Al Davis, Ed Yourdon, Grady Booch, Philippe Kruchten, Leslee Probasco, Ian Spence, Jean Bell, Walker Royce, Joe Marasco, Elemer Magaziner, and the following reviewers of the first edition: Ag Marsonia, Frank Armour, Dr. Ralph R. Young, Professor David Rine, and Dan Rawsthorne.

We admit that without their insightful comments, we could not have written a worthy work. In addition, we'd like to thank Kim Arney Mulcahy and the editors and support staff at Addison-Wesley, who tuned our work in process and helped it become a tangible product, like the "rock" described in the Foreword. Lastly, we must again thank our loving families for supporting all the "heads-down weekends" necessary to complete this second edition.