HZ BOOKS

PEARSON

Addison Wesley

经 典 原 版 书 库

基于用例的面向方面软件开发

(英文版)

ASPECT-ORIENTED SOFTWARE DEVELOPMENT WITH USE CASES

IVAR JACOBSON

PAN-WEI NG

OBJECT TECHNOLOGY SERIES

BOOCH JACOBSON RUMBAUGH

ADDISON-WESLEY

SERIES EDITORS

(美) Ivar Jacobson Pan-Wei Ng 著

机械工业出版社
China Machine Press

经 典 原 版 书 库

基于用例的面向方面软件开发

（英文版）

Aspect-Oriented Software
Development with Use Cases

（美） Ivar Jacobson
Pan-Wei Ng 著

机 械 工 业 出 版 社
China Machine Press

本书版权登记号：图字：01-2005-4833

图书在版编目（CIP）数据

基于用例的面向方面软件开发（英文版）/（美）雅各布森（Jacobson, I.）等著.
-北京：机械工业出版社，2006.1
（经典原版书库）
书名原文：Aspect-Oriented Software Development with Use Cases
ISBN 7-111-17771-1

Ⅰ.基…　Ⅱ.雅…　Ⅲ.软件工程-英文　Ⅳ.TP311.5

中国版本图书馆CIP数据核字（2005）第127463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
北京瑞德印刷有限公司印刷·新华书店北京发行所发行
2006年1月第1版第1次印刷
718mm × 1020mm 1/16 · 28.25印张
印数：0 001- 3 000册
定价：55.00元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换
本社购书热线：（010）68326294

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，有的还专程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：hzjsj@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

专家指导委员会

（按姓氏笔画顺序）

尤晋元	王　珊	冯博琴	史忠植	史美林
石教英	吕　建	孙玉芳	吴世忠	吴时霖
张立昂	李伟琴	李师贤	李建中	杨冬青
邵维忠	陆丽娜	陆鑫达	陈向群	周伯生
周克定	周傲英	孟小峰	岳丽华	范　明
郑国梁	施伯乐	钟玉琢	唐世渭	袁崇义
高传善	梅　宏	程　旭	程时端	谢希仁
裘宗燕	戴　葵			

Praise for *Aspect-Oriented Software Development with Use Cases*

"A refreshingly new approach toward improving use-case modeling by fortifying it with aspect orientation."

—*Ramnivas Laddad*
author of AspectJ in Action

"Since the 1980s, use cases have been a way to bring users into software design, but translating use cases into software has been an art, at best, because user goods often don't respect code boundaries. Now that aspect-oriented programming (AOP) can express crosscutting concerns directly in code, the man who developed use cases has proposed step-by-step methods for recognizing crosscutting concerns in use cases and writing the code in separate modules. If these methods are at all fruitful in your design and development practice, they will make a big difference in software quality for developers and users alike."

—*Wes Isberg*
AspectJ team member

"This book not only provides ideas and examples of what aspect-oriented software development is but how it can be utilized in a real development project."

—*Michael Ward*
ThoughtWorks, Inc.

"No system has ever been designed from scratch perfectly; every system is composed of features layered in top of features that accumulate over time. Conventional design techniques do not handle this well, and over time the integrity of most systems degrades as a result. For the first time, here is a set of techniques that facilitates composition of behavior that not only allows systems to be defined in terms of layered functionality but composition is at the very heart of the approach. This book is an important advance in modern methodology and is certain to influence the direction of software engineering in the next decade, just as *Object-Oriented Software Engineering* influenced the last."

—*Kurt Bittner*
IBM Corporation

"Use cases are an excellent means to capture system requirements and drive a user-centric view of system development and testing. This book offers a comprehensive guide on explicit use-case-driven development from early requirements modeling to design and implementation. It provides a simple yet rich set of guidelines to realize use-case models using aspect-oriented design and programming. It is a valuable resource to researchers and practitioners alike."

—*Dr Awais Rashid*
Lancaster University, U.K., and author of
Aspect-Oriented Database Systems

"AOSD is important technology that will help developers produce better systems. Unfortunately, it has not been obvious how to integrate AOSD across a project's lifecycle. This book shatters that barrier, providing concrete examples on how to use AOSD from requirements analysis through testing."

—*Charles B. Haley*
research fellow, The Open University, U.K.

Preface

What Is Aspect-Oriented Programming?

That you have picked up this book tells us that you are a member of the software development community: a tester, a developer, a project leader, a project manager, an architect, an analyst, or a member involved in one of the many other aspects of developing. We also know that you are someone who wants to improve the way you develop software. You want your system to be more maintainable, more extensible, more reusable, and if you are a project leader, you want your team to be more productive. You know that these goals are not always easy to achieve.

Why is software development so difficult? One reason is that there are many things to watch out for. On the human side, you have to watch out for time, budget, resources, skills, and so forth. Frequently, as a team member, you have many tasks—some of them beyond what you are paid for. You report to two different people and each expects 100 percent from you, so you must give 200 percent to your work. As the developer, you must understand the application, the domain, and the idiosyncrasies of the platform. When you design the system, you need to deal with and balance many difficult concerns: how the system meets its intended functionality, how it achieves performance and reliability, how it deals with platform specifics, and so forth. You may find that your code—your classes, your operations, your procedures—must perform many functions, which may lead to spaghetti code, an indication of poor design. So, you need to improve design—improve modularity and provide better *separation of concerns.* Just as each team member must be clearly focused on his or her work, each component, each class, each operation must be focused on what is its specific purpose.

But there is a limit to what you can do with existing techniques. No matter how far you go, you find that many parts of your system have code fragments that have to do with logging, authorization, persistence, debugging, tracing, distribution, exception handling, and other such tasks. Sometimes, a sizeable portion of an operation or class has nothing to do with what it is supposed to do. Aspect-oriented programming (AOP) refers to such redundancy as *crosscutting concerns* because you find these code fragments in many operations and classes in your system—they *cut across* operations and classes. Crosscutting concerns are not limited to the technical concerns such as authorization and persistence. They include system and application functionality, and you find that a change in functionality often results in changes in many classes too.

AOP gives you the means to separate code that implements crosscutting concerns and modularize it into *aspects.* Aspect-orientation provides the mechanism to compose crosscutting behaviors into the desired operations and classes during compile time and even during execution. The source code for your operations and classes can be free of crosscutting concerns and therefore easier to understand and maintain.

What Is Aspect-Oriented Software Development?

In order to progress beyond AOP, you need a holistic approach to developing software systems with aspects from requirements, to analysis and design, to implementation and test. This is aspect-oriented software development (AOSD).

AOSD is about better modularity for the entire system, encompassing concerns of many different kinds—better modularity for functional requirements, nonfunctional requirements, platform specifics, and so on—and keeping them separate from each other. Keeping all concerns separate allows you to construct systems that have a more understandable structure and are easily configured and extended to meet the evolving needs of stakeholders.

AOSD is not just AOP. It encompasses a whole range of techniques to help you achieve better modularity. These techniques include object orientation, component-based development, design patterns, object-oriented frameworks such as J2EE and .NET, and more. AOSD does not compete with existing techniques but is built on top of them.

AOSD with Use Cases

How do you conduct AOSD? How do you identify aspects? When do you use classes as opposed to aspects? How do you specify aspects? You need a sound and systematic approach to help you conduct AOSD. The development community is crying out for this kind of systematic approach to software development.

In fact, there is such a systematic approach—and a mature one too. It is called the use-case-driven approach. It provides a sound method for developing applications by focusing on realizing stakeholder concerns and delivering value to the user.

It is well known that aspect orientation helps modularize crosscutting concerns during implementation, but there is a need to modularize crosscutting concerns much earlier, even during requirements. Use-cases are an excellent technique for this purpose. Use-cases are crosscutting concerns, since the realization of use cases touches several classes. In fact, you can model *most* crosscutting concerns with use-cases, and we demonstrate use-case modeling in the book.

The underlying concept in aspect orientation is similar to the concept of use-case-driven development. This means that you get a seamless transition from expressing requirements of stakeholder concerns with use-cases to implementing them with aspects.

Briefly, you conduct AOSD with use-cases as follows: You model crosscutting concerns with use-cases. You design use-cases in terms of overlays on top of classes—overlays called use-case slices and use-case modules. You use aspect technology to compose use-cases slices and use-case modules to form the complete model for the system.

We use a home-construction analogy to explain the approach further. Let's say you have a new house, but it is just an empty house with no fixtures—no lights, no phone lines, no wiring, no gas, and no Internet! Each missing fixture or service is a distinct concern, evidenced by the fact that you need to call different specialists to install each fixture or service. The fixtures and services are *crosscutting concerns*—they cut across different rooms (i.e., objects). They are analogous to use-cases. To determine how he or she will go about his or her job, each specialist must design a plan, often in terms of a diagram based on the floor plan. The floor plan shows where the rooms and the walls are. The electrician makes a photocopy of the floor plan and draws how she intends to install electric wiring; the plumber sketches out how he plans to run water pipes around the house; and so on. Each specialist can work separately, but all of them base their work on the same floor plan. The overall work to be done is the sum of all these diagrams.

If each specialist were to draw his or her diagram on a transparency, the transparencies could be merged by overlaying them on a projector. These overlays are analogous to what we call use-case slices and use-case modules. As long as the overlays are based on the same dimensions of the floor plan, you can get a perfect image on the screen showing *all* the work to be done. If there is a need to change the laying of Internet lines, you just rework the overlay that describes that plan and update the merged model. When you project it with the other overlays, you get the updated image of the house. You can easily stack more overlays on the picture or swap in and out overlays. You get a coherent image provided that the dimensions correspond to each other. This represents the architectural work involved.

Systems developed using use-case slices and use-case modules have a clear separation of crosscutting concerns. You can evolve them and extend them. It is easier to make each slice reusable. You can automatically generate certain slices because they do not interfere with other slices. You get better maintainability, better extensibility, and greater productivity with this approach.

The development community can gain even more from conducting AOSD with use-cases. We believe that the adoption of aspect orientation will accelerate significantly by basing it on the use-case-driven approach because this approach has already been widely accepted as a means to drive system development, testing, and delivery. Much literature on the use-case-driven approach is readily available for the development community. A good number of professionals, even companies, exist primarily to instruct and promote its use. Project teams both large and small have been successful in adopting the approach. Thus, it is attractive and even natural to base AOSD on the use-case-driven approach.

What This Book Is

This book systematically outlines how to conduct AOSD with use-cases. We cover requirements, analysis, design, implementation, and test. We demonstrate how to model crosscutting concerns and aspects with UML and how to establish a resilient architecture that is based on use-cases and aspects. We highlight key changes in practice and the paradigm shifts that you must note when applying AOSD. We give pointers on how you can quickly reap the benefits of AOSD in your projects.

We demonstrate how you conduct AOSD in a mixed environment of object-oriented frameworks such as J2EE, object-oriented design patterns, AOP, and so on, because we recognize that these are the challenges you face in practice. We show you how to map aspect and use-case analysis to different design and implementation technologies.

We spend a great deal of time in this book describing how to establish a firm architecture based on use-cases and aspects—an architecture that is resilient to changes.

Some of you may be familiar with earlier works by Ivar Jacobson, such as *Object-Oriented Software Engineering: A Use-Case Driven Approach* (Addison-Wesley, 1992) and *The Unified Software Development Process* (Addison-Wesley, 1999). This book should be read in conjunction with those books.

Some of you may have read books on aspect orientation and wondered about its implications for software development as a whole. This is the book for you. Newcomers to aspect orientation will learn its principles and application. If you are familiar with the use-case-driven approach, you should readily recognize the benefits and implications of aspect orientation. This book will help you to appreciate the larger context of aspects—not just AOP, but AOSD.

In this book, we use a single example of a Hotel Management System, which you become familiar with as we progress through the book. By building upon a single example, we keep our discussion of aspect orientation and use-cases focused and concrete.

What This Book Is Not

This book is not a programming book. We do not go into details about AOP languages or aspect-oriented frameworks that are currently available. For those details, refer to guide books and tutorials. This book is about aspect-oriented *software development* (not just programming). The emphasis is on a software development approach from requirements to code, applying a number of techniques in a balanced and iterative manner to help you succeed in building your software systems.

This book does not attempt to be an aspect-oriented design cookbook. We do not attempt to discuss all conceivable crosscutting concerns (synchronization, transaction management, caching, etc.). Nevertheless, we believe that the breadth of this book provides the principles and the basis for you to deal with many kinds of crosscutting concerns that you will encounter in practice.

What You Need Before Reading This Book

There are several prerequisites to getting the most out of this book. You must have at least some understanding of the Unified Modeling Language (UML). We include some explanation of UML in this book, but we largely expect you to know the basics. We expect you to know what classes are and that you can read use-case diagrams, class diagrams, sequence diagrams, and collaboration diagrams. Incidentally, the last two are called *interaction* diagrams and *communication* diagrams in UML 2.0.

If you have ever applied use-case-driven development in a project, then you will really benefit from this book—even if you do not have any background in aspect orientation. We had you in mind when we wrote this book and spent some time to ground you in the basics of AOP.

If you are knowledgeable about aspect orientation and have little idea about use-case-driven development, do not fret. We have you in mind, too. Part II is devoted to acquainting you with use cases and use-case realizations.

We show some code examples in AspectJ to give you a concrete picture of our proposed extension to the UML notation to support AOSD. AspectJ is a language extension of Java that supports AOP. Some understanding of Java is therefore helpful. However, we want to highlight that this is not an AOP book. This book does not intend to teach you the complete AspectJ syntax.

Since we are showing you how to apply AOSD in a mixed environment and how to deal with platform specifics, we need to use some platform to make our discussion concrete. For this purpose, we chose J2EE, so some knowledge of J2EE is useful. If you have ever heard about servlets and EJBs, you should have sufficient background. If you know the J2EE core patterns, better still.

So, welcome—and read this book.

How to Read This Book

We organized this book into five parts:

Part I, The Case for Use Cases and Aspects

Part I is basically an expansion of this preface. The goal is to help you understand what AOSD with use cases is all about. We highlight some basic cases of crosscutting—peers and extensions—and how aspects solve them. Through some simple code examples, we introduce AspectJ, which is currently the most popular AOP technology. We provide an overview of use-case-driven development as it is today—what use cases are, how use cases are realized, and how they are mapped to classes—and what we expect it to be like tomorrow—with aspects, use case slices and use case modules.

Part II, Modeling and Capturing Concerns with Use Cases

Whether you are familiar with use cases or not, you should read Part II. It gives you an overview of the use-case technique and clarifies common misconceptions about use cases. Part II also enhances the use-case modeling technique to provide a seamless transition from use-case modeling to aspect-oriented analysis and design. In particular, we show how pointcuts are modeled within use cases. Part II culminates by walking through a rich example of applying use-case modeling to different crosscutting concerns—both functional and nonfunctional. They are modeled with use cases of different kinds—application use cases and infrastructure use cases. Subsequent parts of the book demonstrate how to drive these different kinds of use cases all the way to implementation.

Part III, Keeping Concerns Separate with Use Case Modules

Part III goes deeper into the concept of use-case slices and use-case modules. Use-case slices help you keep the specifics of a use case separate in the design model. They are the overlays we mentioned to keep crosscutting concerns separate. We show you how to model use-case slices and aspects with UML and how our extensions to the UML notation map to AOP. We use this notation and the underlying concepts in Part IV, and the notation is summarized in Appendix B.

Part IV, Establishing an Architecture Based on Use Cases and Aspects

The most important determinant to the success of a project lies in its architecture. Part IV demonstrates how to get good architecture, step by step. Among other things, a good architecture keeps concerns of different kinds separate. It separates the use-case-specific from the use-case-generic, the application-specific from the application-generic; the platform-specific from the platform-independent; and so on. This separation not only makes your system more understandable and maintainable, it also makes your system extensible. It makes parts of your system reusable without causing those parts to be entangled. It also provides room for significant automation when implementing the system.

In Part IV, there are plenty of useful tips and guidelines to such an architecture. You might need to refer to Appendix B on the notations used in the book as you read Part IV.

Part V, Applying Use Cases and Aspects in a Project

You might be at the beginning, middle, or final stages of your project: No matter what stage you are at, you can apply the practices advocated in this book. In Part V, we demonstrate how to tailor our approach to different project scenarios. We also show you how to effectively manage a project that is based on use cases and aspects.

How This Book Came About

Research papers often have many authors, and you may wonder what each author contributed to the work. Here, we reminisce about how this book came about and explain our individual contributions to the concepts and pragmatics described in this book.

In the Beginning
By Ivar

The first time I heard the term aspect-oriented programming was back in 1997. I immediately saw it as an interesting technology, but at the time, I couldn't take a serious look at. I was working on the first version of UML, and on getting Rational Unified Process (RUP) right, and I was initiating an effort on active software processes—actually what is now Jaczone WayPointer. When I finally had time to take a good look at aspects, it was in September 2002. I downloaded a lot of papers and studied them for a couple of days. Then I contacted Harold Ossher at IBM Research and Karl Lieberherr at Northeastern University in Boston. They are two of the leaders in this space. The most well-known person on aspects is Gregor Kizcales. I tried to get his attention as well, but he was too busy at that time. However, he contributed to this book by reviewing it extensively, and we benefited significantly from his comments.

In November 2002, I visited the IBM folks and spent a day with them learning about their work. After the meeting, I was very impressed and excited about what they had done. I left their office and rushed to Newark airport; I had to run to the gate. This is normal. I was on my way to Stockholm. When I was seated in the plane, I ordered some champagne and began to relax and think a little. Suddenly, it struck me. Didn't I do something similar before? Didn't I write a paper on the same subject for OOPSLA '86—the very first OOPSLA conference?

When I got to Stockholm, I hunted for the paper. It was a paper that discussed a topic that I mentioned as future work in my Ph.D. thesis from 1985. However, the ideas in the paper had generated no interest, so I decided to leave the subject. I felt it was too early to push those ideas and just forgot about them. Besides, my work on component-based development with objects and use cases was so successful that there was no room for new inventions. However, now I wanted to find the paper. I went to the publisher's Web site and found it. I had to pay $95 to download it! My own paper!

The title of the paper is "Language Support for Changeable Large Real-Time Systems" [Jacobson 1986]. In that paper, I introduced several concepts—existion, which represents a base, and extension, which represents separate functionality that you want to add to the existion. Instead of modifying the existion to invoke an extension, we used a mechanism to allow the extension to add behavior into the existion. Thus, from the perspective of the existion, no modification was needed. This meant that you could add extension after extension without breaking the existion. The key idea is this: By keeping extensions separate from the base from the developer's perspective, the system is much easier to understand, maintain, and extend.

The basic idea sounded very much like what aspect-orientation research is trying to achieve. But I needed confirmation. Two hours after I downloaded the paper, I sent it to Karl Lieberherr. He responded, "Wow, Ivar, this is an early paper on aspect-orientation." He asked me if I had anything more. Since I throw away everything I don't work with, my first thought was that there was nothing more. However, I was excited, and my thoughts went back to the time before the paper. My memory asked me, "Didn't you file a patent for a similar work?"