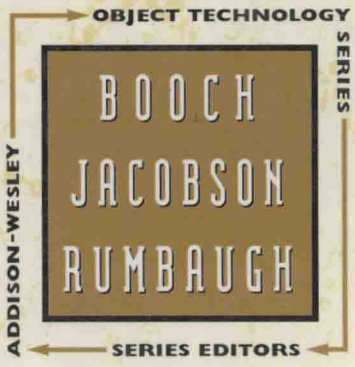
UML与面向对象设计影印丛书

面向对象系统测试
模型 视图与工具

TESTING OBJECT-ORIENTED SYSTEMS
MODELS, PATTERNS, AND TOOLS

ROBERT V. BINDER 编著



科学出版社
www.sciencep.com

(TP-1954.0101)
Software Testing/Object Technology

More than ever, mission-critical and business-critical applications depend on object-oriented (OO) software. Testing techniques tailored to the unique challenges of OO technology are necessary to achieve high reliability and quality. ***Testing Object-Oriented Syetems: Model, Patterns, and Tools*** is an authoritative guide to designing and automating test suites for OO applications.

This comprehensive book explains why testing must be model-based and provides in-depth coverage of techniques to develop testable models from state machines, combinational logic, and the Unified Modeling Language (UML). It introduces the test design pattern and presents 37 patterns that explain how to design responsibility-based test suites, how to tailor integration and regression testing for OO code, how to test reusable components and frameworks, and how to develop highly effective test suites from use cases.

Effective testing must be automated and must leverage object technology. The author describes how to design and code specification-based assertions to offset testability losses due to inheritance and polymorphism. Fifteen micro-patterns present oracle strategies--practical solutions for one of the hardest problems in test design. Seventeen design patterns explain how to automate your test suites with a coherent OO test harness framework

The author provides thorough coverage of testing issues such as:

- The bug hazards of OO programming and differences from testing procedural code
- How to design responsibility-based tests for classes, clusters, and subsystems using class invariants, interface data flow models, hierarchic state machines, class associations, and scenario analysis
- How to supper reuse by effective testing of abstract classes, generic classes, components, and frameworks
- How to choose an integration strategy that supports iterative and incremental development
- How to achieve comprehensive system testing with testable use cases
- How to choose a regression test approach
- How develop expected test results and evaluate the post-test state of an object
- How automate testing with assertions, OOtest drivers,stubs,and test frameworks

Real-world experience, world-class best practices, and the latest research in object-oriented testing are included. Practical examples illustrate test design and test automation for Ada 95, C++, Eiffel, Java, Objective-C, and Smalltalk. The UML is used throughout, but the test design patterns apply to systems developed with any OO language or methodology.

Robert V. Binder, president and founder of RBSC Corporation, is internationally recognized as the leading expert in testing object-oriented systems. With more than 25 years of software development experience in a wide range of technical and management roles, he has implemented advanced OO test design and automation solutions for hundreds of clients. He is the author of Application Debugging, writes a column on testing for Component Strategies, and has published many articles in peer-reviewed and popular journals. He serves on the board of the annual Quality Week conference and is a senior member of the IEEE.

Find more information about the **Object Technology Series** at http://www.awl.com/cseng/otseries

ADDISON-WESLEY
Pearson Education
http://www.PearsonEd.com

ISBN 7-03-011399-3

ISBN 7-03-011399-3
定 价：120.00 元

面向对象系统测试模型 视图与工具

社

UML与面向对象设计影印丛书

面向对象系统测试
模型 视图与工具

Robert V. Binder 编著

科学出版社

内 容 简 介

要实现高质量、高稳定性的面向对象软件系统，有效的测试技术是必不可少的。本书深入讲述了如何用状态机、组合逻辑和 UML 开发可测试的模型。通过对多种模式的介绍，可以让读者掌握如何设计测试套件、如何针对 OO 代码修改测试方法、如何测试可重用组件及框架，以及如何根据用况开发高效的测试套件。书中还提供了许多实际测试经验和面向对象测试领域的最新研究成果。

与面向对象系统稳定性有关的开发及测试人员，皆可阅读此书。

ISBN 0-201-80938-9

图字：01-2003-2540

图书在版编目(CIP)数据

面向对象系统测试：模型、视图与工具=Testing Object-Oriented Systems: Models, Patterns and Tools/（美）宾德（Binder,R.V.）著.—影印本. —北京：科学出版社，2003

ISBN 7-03-011399-3

Ⅰ.面… Ⅱ.宾… Ⅲ.面向对象语言—程序设计—英文 Ⅳ.TP312

中国版本图书馆 CIP 数据核字（2003）第 030826 号

策划编辑：李佩乾/责任编辑：李佩乾
责任印制：吕春珉/封面制作：东方人华平面设计室

科学出版社 出版
北京东黄城根北街16号
邮政编码:100717
http://www.sciencep.com

双青印刷厂 印刷
科学出版社发行 各地新华书店经销

*

2003 年 5 月第 一 版 开本：787×960 1/16
2003 年 5 月第一次印刷 印张：77 1/2
印数：1—2 000 字数：1 483 000

定价：120.00 元

（如有印装质量问题，我社负责调换<环伟>）

影印前言

随着计算机硬件性能的迅速提高和价格的持续下降，其应用范围也在不断扩大。交给计算机解决的问题也越来越难，越来越复杂。这就使得计算机软件变得越来越复杂和庞大。20 世纪 60 年代的软件危机使人们清醒地认识到按照工程化的方法组织软件开发的必要性。于是软件开发方法从 60 年代毫无工程性可言的手工作坊式开发，过渡到 70 年代结构化的分析设计方法、80 年代初的实体关系开发方法，直到面向对象的开发方法。

面向对象的软件开发方法是在结构化开发范型和实体关系开发范型的基础上发展而来的，它运用分类、封装、继承、消息等人类自然的思维机制，允许软件开发者处理更为复杂的问题域和其支持技术，在很大程度上缓解了软件危机。面向对象技术发端于程序设计语言，以后又向软件开发的早期阶段延伸，形成了面向对象的分析和设计。

20 世纪 80 年代末 90 年代初，先后出现了几十种面向对象的分析设计方法。其中，Booch, Coad/Yourdon、OMT 和 Jacobson 等方法得到了面向对象软件开发界的广泛认可。各种方法对许多面向对象的概念的理解不尽相同，即便概念相同，各自技术上的表示法也不同。通过 90 年代不同方法流派之间的争论，人们逐渐认识到不同的方法既有其容易解决的问题，又有其不容易解决的问题，彼此之间需要进行融合和借鉴；并且各种方法的表示也有很大的差异，不利于进一步的交流与协作。在这种情况下，统一建模语言(UML)于 90 年代中期应运而生。

UML 的产生离不开三位面向对象的方法论专家 G. Booch、J. Rumbaugh 和 I. Jacobson 的通力合作。他们从多种方法中吸收了大量有用的建模概念，使 UML 的概念和表示法在规模上超过了以往任何一种方法，并且提供了允许用户对语言做进一步扩展的机制。UML 使不同厂商开发的系统模型能够基于共同的概念，使用相同的表示法，呈现彼此一致的模型风格。1997 年 11 月 UML 被 OMG 组织正式采纳为标准的建模语言，并在随后的几年中迅速地发展为事实上的建模语言国际标准。

UML 在语法和语义的定义方面也做了大量的工作。以往各种关于面向对象方法的著作通常是以比较简单的方式定义其建模概念，而以主要篇幅给出过程指导，论述如何运用这些概念来进行开发。UML 则以一种建模语言的姿态出现，使用语言学中的一些技术来定义。尽管真正从语言学的角度看它还有许多缺陷，但它在这方面所做的努力却是以往的各种建模方法无法比拟的。

从 UML 的早期版本开始，便受到了计算机产业界的重视，OMG 的采纳和大公司的支持把它推上了实际上的工业标准的地位，使它拥有越来越多的用户。它被广泛地用

于应用领域和多种类型的系统建模，如管理信息系统、通信与控制系统、嵌入式实时系统、分布式系统、系统软件等。近几年还被运用于软件再工程、质量管理、过程管理、配置管理等方面。而且它的应用不仅仅限于计算机软件，还可用于非软件系统，例如硬件设计、业务处理流程、企业或事业单位的结构与行为建模，等等。

在 UML 陆续发布的几个版本中，逐步修正了前一个版本中的缺陷和错误。即将发布的 UML2.0 版本将是对 UML 的又一次重大的改进。将来的 UML 将向着语言家族化、可执行化、精确化等理念迈进，为软件产业的工程化提供更有力的支撑。

本丛书收录了与面向对象技术和 UML 有关的 12 本书，反映了面向对象技术最新的发展趋势以及 UML 的新的研究动态。其中涉及对面向对象建模理论研究与实践的有这样几本书：《面向对象系统架构及设计》主要讨论了面向对象的基本概念、静态设计、永久对象、动态设计、设计模式以及体系结构等近几年来面向对象技术领域中的新的理论知识与方法；《用 UML 进行用况对象建模》主要介绍了面向对象的需求阶段、分析阶段、设计阶段中用况模型的建立方法与技术；《高级用况建模》介绍了在建立用况模型中需要注意的高级的问题与技术；《UML 面向对象设计基础》则侧重于经典的面向对象理论知识的阐述。

涉及 UML 在特定领域的运用的有这样几本：《UML 实时系统开发》讨论了进行实时系统开发时需要对 UML 进行扩展的技术；《用 UML 构建 Web 应用程序》讨论了运用 UML 进行 Web 应用建模所应该注意的技术与方法；《面向对象系统测试：模型、视图与工具》介绍了将 UML 应用于面向对象的测试领域所应掌握的方法与工具；《对象、构件、框架与 UML 应用》讨论了如何运用 UML 对面向对象的新技术——构件-框架技术建模的方法策略。《UML 与 Visual Basic 应用程序开发》主要讨论了从 UML 模型到 Visual Basic 程序的建模与映射方法。

介绍面向对象编程技术的有两本书：《COM 高手心经》和《ATL 技术内幕》，深入探讨了面向对象的编程新技术——COM 和 ATL 技术的使用技巧与技术内幕。

还有一本《Executable UML 技术内幕》，这本书介绍了可执行 UML 的理念与其支持技术，使得模型的验证与模拟以及代码的自动生成成为可能，也代表着将来软件开发的一种新的模式。

总之，这套书所涉及的内容包含了对软件生命周期的全过程建模的方法与技术，同时也对近年来的热点领域建模技术、新型编程技术作了深入的介绍，有些内容已经涉及到了前沿领域。可以说，每一本都很经典。

有鉴于此，特向软件领域中不同程度的读者推荐这套书，供大家阅读、学习和研究。

北京大学计算机系 蒋严冰 博士

Thus spake the master: "Any program, no matter how small, contains bugs."

The novice did not believe the master's words. "What if the program were so small that it performed a single function?" he asked.

"Such a program would have no meaning," said the master, "but if such a one existed, the operating system would fail eventually, producing a bug."

But the novice was not satisfied. "What if the operating system did not fail?" he asked.

"There is no operating system that does not fail," said the master, "but if such a one existed, the hardware would fail eventually, producing a bug."

The novice still was not satisfied. "What if the hardware did not fail?" he asked.

The master gave a great sigh. "There is no hardware that does not fail," he said, "but if such a one existed, the user would want the program to do something different, and this too is a bug."

A program without bugs would be an absurdity, a nonesuch. If there were a program without any bugs then the world would cease to exist.

Geoffrey James
The Zen of Programming

Foreword

Some early enthusiastic but misguided advocates of object-oriented programming (OOP) dismissed testing in the erroneous belief that the adoption of OOP would so reduce the incidence of bugs that testing would no longer be needed. We first heard similar claims two generations ago in the context of adopting Cobol as a standard programming language. More recently, CASE failed to deliver on its promise despite clear productivity advantages. For all three, Cobol, CASE, and lately OO, if adoption of the paradigm were to increase productivity to the point where there would be no labor in code creation, all that would be left would be testing and debugging—consuming 100 percent of the labor content. As was learned over decades of sometimes bitter experience for procedural programming languages, every advance has a price. In the case of OO, the very things that lead to greater flexibility, robustness, generality, and productivity are also the things that conspire to make testing, if not more difficult, then at least more challenging.

Nearly everything we have learned about testing procedural language programs also applies to testing OO implementations. Object-oriented testing, as exposited in this book, is built on that infrastructure. However, the emphasis and effectiveness of various test techniques is different for OO. For example, one might never have reason to use either dataflow testing or finite-state machine testing for an application written in a procedural programming language: for an application that exploits what OOP has to offer, the use of these techniques is inescapable. In addition, the relative emphasis on unit and integration testing changes. In procedural languages, unit testing is of primary importance and integration testing is secondary. In OOP, the relative importance is reversed.

Object-oriented programming also brings new problems for testers, problems that are not to be found in procedural programming. Of these, polymorphism, inheritance, and dynamic binding are the most problematic—and they are at the heart of OO. Some of the early research on OO testing was distinctly

pessimistic—going so far as to say "What's the use of OO? We can never test it properly, and probably never really debug it." Both the research community and astute practitioners of OOP were not willing to accept that. What has emerged from those communities' mutual concerns is an approach to testing OO software that uses new techniques and/or old techniques reworked to fit the new paradigm. This knowledge, however, for the most part, has been inaccessible to the practitioner; it lay scattered among hundreds of research papers or in the largely unpublished folklore of OOP. Binder has rectified this gap in a skillful exposition of research results tempered by the harsh realities of practice in an edifice that provides methods and techniques for OOP, while building on a solid foundation of what has been proven through decades of use in previous programming paradigms. This book, I believe, provides the missing half of OOP—the testing half.

Boris Beizer
Abington, Pennsylvania

Preface

What Is This Book About?

Testing Object-Oriented Systems is a guide to designing test suites and test automation for object-oriented software. It shows how to design test cases for any object-oriented programming language and object-oriented analysis/design (OOA/D) methodology. Classes, class clusters, frameworks, subsystems, and application systems are all considered. Practical and comprehensive guidance is provided for many test design questions, including the following:

- How to design responsibility-based tests for classes and small clusters using behavior models, state-space coverage, and interface dataflow analysis.
- How to use coverage analysis to assess test completeness.
- How to design responsibility-based tests for large clusters and subsystems using dependency analysis and hierarchic state models.
- How to design responsibility-based tests for application systems using OOA/D models.
- How to automate test execution with object-oriented test drivers, stubs, test frameworks, and built-in test.

This book is about systems engineering and software engineering as much as it is about testing object-oriented software. *Models* are necessary for test design—this book shows you how to develop testable models focused on preventing and removing bugs. *Patterns* are used throughout to express best practices for designing test suites. *Tools* implement test designs—this book shows you how to design effective test automation frameworks.

Is This Book for You?

This book is intended for anyone who wants to improve the dependability of object-oriented systems. The approaches presented range from basic to advanced. I've tried to make this book like a well-designed kitchen. If all you want is a sandwich and a cold drink, the high-output range, large work surfaces, and complete inventory of ingredients won't get in your way. But the capacity is there for efficient preparation of a seven-course dinner for 20 guests, when you need it.

I assume you have at least a working understanding of object-oriented programming and object-oriented analysis/design. If you're like most OO developers, you've probably specialized in one language (most likely C++ or Java) and you may have produced or used an object model. I don't assume that you know much about testing. You will need some background in computer science and software engineering to appreciate the advanced material in this book, but you can apply test design patterns without specialized theoretical training.

You'll find this book useful if you must answer any of the following questions.

- What are the differences between testing procedural and object-oriented software?
- I've just written a new subclass and it seems to be working. Do I need to retest any of the inherited superclass features?
- What kind of testing is needed to be sure that a class behaves correctly for all possible message sequences?
- What is a good integration test strategy for rapid incremental development?
- How can models represented in the UML be used to design tests?
- What can I do to make it easier to test my classes and applications?
- How can I use testing to achieve greater reuse?
- How should I design test drivers and stubs?
- How can I make my test cases reusable?
- How can I design a good system test plan for an OO application?
- How much testing is enough?

The material here is not limited to any particular OO programming language, OOA/D methodology, kind of application, or target environment. How-

ever, I use the Unified Modeling Language (UML) throughout. Code examples are given in Ada 95, C++, Java, Eiffel, Objective-C, and Smalltalk.

A Point of View

My seven-year-old son David asked, "Dad, why is your book so big?" I'd just told David that I'd have to leave his baseball game early to get back to work on my book. I wanted to explain my choice, so I tried to be clear and truthful in answering. This is what I told David at the kitchen table on that bright summer afternoon:

> Testing is complicated and I'm an engineer. Making sure that things work right is very important for engineers. What do you think would happen if our architect didn't make our house strong enough because he was lazy? It would fall down and we could get hurt. Suppose the engineers at GM did only a few pages' worth of testing on the software for the brakes in our car. They might not work when we need them and we'd crash. So when engineers build something or answer a question about how to build things, we have to be sure we're right. We have to be sure nothing is left out. It takes a lot of work.

As I was speaking, I realized this was the point of view I'd been struggling to articulate. It explains why I wrote this book and the way I look at the problem of testing object-oriented software. Testing is an integral part of software engineering. Object-oriented technology does not diminish the role of testing. It does alter some important technical details, compared with other programing paradigms. So, this is a large book about how testing, viewed as software engineering, should be applied to object-oriented systems development. It is large because testing and object-oriented development are both large subjects, with a large intersection. By the way—David hit two home runs later that afternoon while I was torturing the truth out of some obscure notions.

Acknowledgments

No one who helped me with this book is responsible for its failings.[1] Dave Bulman, Jim Hanlon, Pat Loy, Meilir Page-Jones, and Mark Wallace reviewed the first technical report about the FREE methodology [Binder 94].

In 1993, Diane Crawford, editor of *Communications of the ACM,* accepted my proposal for a special issue on object-oriented testing, which was published in September 1994. The contributors helped to shape my views on the relationship between the development process and testing. Bill Sasso (then with Andersen Consulting and now answering a higher calling) sponsored a presentation where questions were asked that led to development of the Mode Machine Test pattern (see Chapter 12). Bob Ashenhurst of the University of Chicago, James Weber, and the rest of the Regis Study Group raised more fundamental questions: What is a state? Why should we care about pictures?

The following year, Marie Lenzie, as editor of *Object Magazine,* accepted my proposal for a bimonthly testing column. Since 1995, writing this column has forced me to transform often hazy notions into focused, pragmatic guidance six times each year. Lee White of CASE Western Reserve University and Martin Woodward of the University of Liverpool, editors of the journal *Software Testing, Verification, and Reliability,* encouraged my work in developing a comprehensive survey, patiently waited, and then allocated an entire issue to its publication. Writing the survey helped to sort which questions were important, why they were asked, and what the best available thinking did and did not answer.

My publications, conference tutorials, and professional development seminars on object-oriented testing served as a conceptual repository and proving ground. Many of these materials, with the necessary changes, have been reused here. The cooperation of RBSC Corporation, SIGS Publications, the ACM, the IEEE, and Wiley (U.K.) is appreciated in this regard (see Sources and Credits

1. John Le Carre crafted this concise statement about assistance he received on *The Tailor of Panama.* I can't improve on it.

that follow for details). The real-world problems and questions posed by my consulting clients and thousands of seminar participants have been humbling and constant spurs to refinement.

The patient support of Carter Shanklin and his predecessors at Addison-Wesley kept this project alive. Boris Beizer's steady encouragement, suggestions, and acerbic critiques have been invaluable.

Several adept programmers suggested code examples or helped to improve my own: Brad Appleton (C++ in the ***Percolation*** pattern and elsewhere), Steve Donelow (Objective-C built-in test), Dave Hoag (Java inner class drivers), Paul Stachour (Ada 95 assertions and drivers), and Peter Vandenberk (Objective-C assertions).

Drafts of patterns, chapters, and the entire book have been reviewed by many people. I am very grateful for the reviewers' thoughtful and detailed feedback. Elaine Weyuker helped to debug my interpretation of her Variable Negation strategy presented in Chapter 6. Brad Appleton and the Chicago Patterns Study Group held two pattern writer's workshops that focused on the test design pattern template and early versions of the ***Invariant Boundary*** and ***Percolation*** patterns. Ward Cunningham commented on an early draft of the test pattern template. Several people reviewed test patterns based on their work: Tom Ostrand (Category-Partition), John Musa (Allocate Tests by Profile), and Michael Feathers (Incremental Testing Framework). Derek Hatley reviewed an early version of Combinational Logic (Chapter 6); Lee White, Regression Testing (Chapter 15); Doug Hoffman, Oracles (Chapter 18); and Dave Hoag, Test Harness Design (Chapter 19). Anonymous reviewers of an early version of the manuscript pointed out many opportunities for improvement. Brad Appleton, Boris Beizer, Camille Bell, Jim Hanlon, and Paul Stachour reviewed the entire final manuscript and provided highly useful commentary.

Finally, thanks to Judith, Emily, and David for years of support, patience, and encouragement.

Sources and Credits

Some of the author's previous publications have been reused or adapted under the terms of the copyright agreements with original publishers of *Object Magazine, Component Strategies, Communications of the ACM,* and the *Journal of Software Testing, Verification and Reliability.* See the Bibliographic Notes section in each chapter for specific citations.