

<<软件测试基础>>

图书基本信息

书名：<<软件测试基础>>

13位ISBN编号：9787111282464

10位ISBN编号：7111282469

出版时间：2009-10

出版时间：机械工业出版社

作者：（美）阿曼，奥法特 著

页数：308

版权说明：本站所提供下载的PDF图书仅提供预览和简介，请支持正版图书。

更多资源请访问：<http://www.tushu007.com>

前言

This book presents software testing as a practical engineering activity , essential to producing high-quality software . It is designed to be used as the primary textbook in either an undergraduate or graduate course on software testing , as a supplement to a general course on software engineering or data structures , and as a resource for software test engineers and developers . This book has a number of unique features : It organizes the complex and confusing landscape of test coverage criteria with a novel and extremely simple structure . At a technical level , software testing is based on satisfying coverage criteria . The book ' S central observation is that there are few truly different coverage criteria , each of which fits easily into one of four categories : graphs , logical expressions , input space , and syntax structures . This not only simplifies testing , but it also allows a convenient and direct theoretical treatment of each category . This approach contrasts strongly with the traditional view of testing , which treats testing at each phase in the development process differently . It is designed and written to be a textbook . The writing style is direct . It builds the concepts from the ground up with a minimum of required background , and it includes lots of examples , homework problems , and teaching materials . It provides a balance of theory and practical application , presenting testing as a collection of objective , quantitative activities that can be measured and repeated . The theoretical concepts are presented when needed to support the practical activities that test engineers follow . It assumes that testing is part of a mental discipline that helps all IT professionals develop higher-quality software . Testing is not an anti-engineering activity , and it is not an inherently destructive process . Neither is it only for testing specialists or domain experts who know little about programming or math . It is designed with modular , interconnecting pieces ; thus it can be used in multiple courses . Most of the book requires only basic discrete math and introductory programming , and the parts that need more background are clearly marked .

<<软件测试基础>>

内容概要

本书采用创新方法来讲述软件测试，定义测试为将几个通用的测试准则应用于软件结构或软件模型的过程。

书中融入了最新的测试技术，包括现代软件方法（如面向对象），Web应用程序和嵌入式软件。另外，本书包含了大量的实例。

作者简介

作者：(美国)阿曼(Paul Ammann) (美国)奥法特(Jeff Offutt)Paul Ammann，在美国维吉尼亚大学获得计算机科学博士学位，现为乔治·梅森大学软件工程副教授。

他于2007年获得乔治·梅森大学Volgenau信息技术与工程学院的杰出教学奖。

Jeff Offutt，在乔治亚理工学院获得计算机博士学位，现为乔治·梅森大学软件工程教授。

他是（（Journal of Software Testing, Verification and Reliability））的主编，是IEEE软件测试、验证和确认国际会议指导委员会主席，还是许多期刊的编委。

他于2003年获得乔治·梅森大学Volgenau信息技术与工程学院的优秀教师奖。

<<软件测试基础>>

书籍目录

Preface	Part 1 Overview	1 Introduction	1.1 Activities of a Test Engineer	1.1.1 Testing Levels Based on Software Activity	1.1.2 Beizer's Testing Levels Based on Test Process Maturity	1.1.3 Automation of Test Activities	1.2 Software Testing Limitations and Terminology	1.3 Coverage Criteria for Testing	1.3.1 Infeasibility and Subsumption	1.3.2 Characteristics of a Good Coverage Criterion	1.4 Older Software Testing Terminology	1.5 Bibliographic Notes
	2 Graph Coverage	2.1 Overview	2.2 Graph Coverage Criteria	2.2.1 Structural Coverage Criteria	2.2.2 Data Flow Criteria	2.2.3 Subsumption Relationships among Graph Coverage Criteria	2.3 Graph Coverage for Source Code	2.3.1 Structural Graph Coverage for Source Code	2.3.2 Data Flow Graph Coverage for Source Code	2.4 Graph Coverage for Design Elements	2.4.1 Structural Graph Coverage for Design Elements	2.4.2 Data Flow Graph Coverage for Design Elements
	2.5 Graph Coverage for Specifications	2.5.1 Testing Sequencing Constraints	2.5.2 Testing State Behavior of Software	2.6 Graph Coverage for Use Cases	2.6.1 Use Case Scenarios	2.7 Representing Graphs Algebraically	2.7.1 Reducing Graphs to Path Expressions	2.7.2 Applications of Path Expressions	2.7.3 Deriving Test Inputs	2.7.4 Counting Paths in a Flow Graph and Determining Max Path Length	2.7.5 Minimum Number of Paths to Reach All Edges	2.7.6 Complementary Operations Analysis
	2.8 Bibliographic Notes	3 Logic Coverage	3.1 Overview: Logic Predicates and Clauses	3.2 Logic Expression Coverage Criteria	3.2.1 Active Clause Coverage	3.2.2 Inactive Clause Coverage	3.2.3 Infeasibility and Subsumption	3.2.4 Making a Clause Determine a Predicate	3.2.5 Finding Satisfying Values	3.3 Structural Logic Coverage of Programs	3.3.1 Predicate Transformation Issues	3.4 Specification-Based Logic Coverage
	3.5 Logic Coverage of Finite State Machines	3.6 Disjunctive Normal Form Criteria	3.7 Bibliographic Notes	4 Input Space Partitioning	4.1 Input Domain Modeling	4.1.1 Interface-Based Input Domain Modeling	4.1.2 Functionality-Based Input Domain Modeling	4.1.3 Identifying Characteristics	4.1.4 Choosing Blocks and Values	4.1.5 Using More than One Input Domain Model	4.1.6 Checking the Input Domain Model	4.2 Combination Strategies Criteria
	4.3 Constraints among Partitions	4.4 Bibliographic Notes	5 Syntax-Based Testing	Part 3 Applying Criteria in Practice	6 Practical Considerations	7 Engineering Criteria for Technologies	8 Building Testing Tools	9 Challenges In Testing Software	List of Criteria	Bibliography	Index	

章节摘录

插图：In Level 1 testing, the purpose is to show correctness: While a significant step up from the naive level 0, this has the unfortunate problem that in any but the most trivial of programs, correctness is virtually impossible to either achieve or demonstrate. Suppose we run a collection of tests and find no failures. What do we know? Should we assume that we have good software or just bad tests?

Since the goal of correctness is impossible, test engineers usually have no strict goal, real stopping rule, or formal test technique. If a development manager asks how much testing remains to be done, the test manager has no way to answer the question. In fact, test managers are in a powerless position because they have no way to quantitatively express or evaluate their work. In Level 2 testing, the purpose is to show failures. Although looking for failures is certainly a valid goal, it is also a negative goal. Testers may enjoy finding the problem, but the developers never want to find problems—they want the software to work (level 1 thinking is natural for the developers). Thus, level 2 testing puts testers and developers into an adversarial relationship, which can be bad for team morale. Beyond that, when our primary goal is to look for failures, we are still left wondering what to do if no failures are found. Is our work done?

Is our software very good, or is the testing weak?

Having confidence in when testing is complete is an important goal for all testers. The thinking that leads to Level 3 testing starts with the realization that testing can show the presence, but not the absence, of failures. This lets us accept the fact that whenever we use software, we incur some risk. The risk may be small and the consequences unimportant, or the risk may be great and the consequences catastrophic, but risk is always there. This allows us to realize that the entire development team wants the same thing—to reduce the risk of using the software. In level 3 testing, both testers and developers work together to reduce risk. Once the testers and developers are on the same “team,” an organization can progress to real Level 4 testing. Level 4 thinking defines testing as a mental discipline that increases quality. Various ways exist to increase quality, of which creating tests that cause the software to fail is only one. Adopting this mindset, test engineers can become the technical leaders of the project (as is common in many other engineering disciplines). They have the primary responsibility of measuring and improving software quality, and their expertise should help the developers. An analogy that Beizer used is that of a spell checker. We often think that the purpose of a spell checker is to find misspelled words,

but in fact, the best purpose of a spell checker is to improve our ability to spell. Every time the spell checker finds an incorrectly spelled word, we have the opportunity to learn how to spell the word correctly. The spell checker is the “expert” on spelling quality. In the same way, level 4 testing means that the purpose of testing is to improve the ability of the developers to produce high quality software. The testers should train your developers. As a reader of this book, you probably start at level 0, 1, or 2. Most software developers go through these levels at some stage in their careers. If you work in software development, you might pause to reflect on which testing level describes your company or team. The rest of this chapter should help you move to level 2 thinking, and to understand the importance of level 3.

<<软件测试基础>>

媒体关注与评论

“关于测试的书很多，但是大部分书涉及的主题范围都很窄并且讲述不详细。而Ammann和Offutt的这本书中所展示的概念和技术广泛地覆盖了业界和学术界使用的各种语言及平台，是一本全面、实用的测试书。

” ——Roger Alexander，华盛顿州立大学

版权说明

本站所提供下载的PDF图书仅提供预览和简介，请支持正版图书。

更多资源请访问:<http://www.tushu007.com>