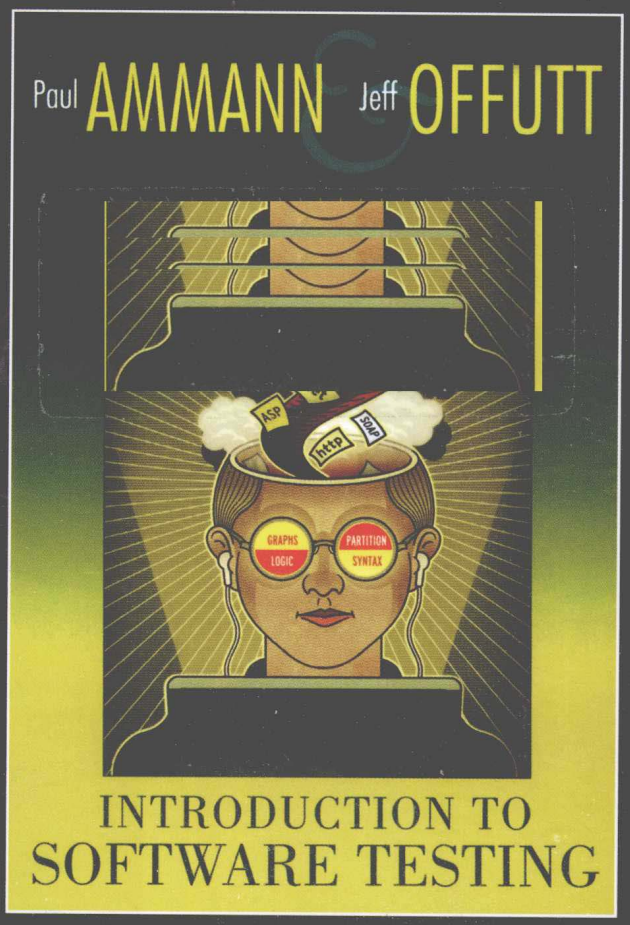
HZ BOOKS
华章教育

计 算 机 科 学 丛 书

CAMBRIDGE

软件测试基础

(美) Paul Ammann 乔治·梅森大学 Jeff Offutt 著
郁莲 北京大学 等译



Introduction to Software Testing

机械工业出版社
China Machine Press

计 算 机 科 学 丛 书

软件测试基础

(美) Paul Ammann Jeff Offutt 著 郁 莲 等译
乔治·梅森大学

Introduction to Software Testing

机械工业出版社
China Machine Press

本书经过了大量的课堂检验，是深受学生和行业专业人员欢迎的软件工程指南。本书所展示的软件测试程概念和技术广泛地覆盖了各种语言及其平台。与其他软件工程书籍相比，本书内容更加全面，并具有很大的实践价值。

本书适合作为国内高等院校计算机及相关专业本科生的软件工程课程教材，也可供软件工程领域的技术人员参考。

Introduction to Software Testing (ISBN 978-0-521-88038-1) by Paul Ammann and Jeff Offutt first published by Cambridge University Press 2008.

本书版权登记号：图字：01-2009-2583

图书在版编目（CIP）数据

软件测试基础 /（美）阿曼（Ammann, P.），奥法特（Offutt, J.）著；郁莲等译. —北京：机械工业出版社，2010.9

（计算机科学丛书）

书名原文：Introduction to Software Testing

ISBN 978-7-111-29398-9

Ⅰ.软…　Ⅱ.①阿…　②奥…　③郁…　Ⅲ.软件—测试　Ⅳ.TP311.5

中国版本图书馆CIP数据核字（2009）第237911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：李俊竹

三河市明辉印装有限公司印刷

2010年10月第1版第1次印刷

184mm×260mm · 16.5印张

标准书号：ISBN 978-7-111-29398-9

定价：36.00元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

客服热线：（010）88378991；88361066

购书热线：（010）68326294；88379649；68995259

投稿热线：（010）88379604

读者信箱：hzjsj@hzbook.com

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，还揭示了学术的源变，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和发展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到积极的推动作用，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分社较早意识到“出版要为教育服务”。自1998年开始，华章分社就将工作重点放在了遴选、移译国外优秀教材上。经过多年的不懈努力，我们与Pearson，McGraw-Hill，Elsevier，MIT，John Wiley & Sons，Cengage等世界著名出版公司建立了良好的合作关系，从他们现有的数百种教材中甄选出Andrew S. Tanenbaum，Bjarne Stroustrup，Brain W. Kernighan，Dennis Ritchie，Jim Gray，Afred V. Aho，John E. Hopcroft，Jeffrey D. Ullman，Abraham Silberschatz，William Stallings，Donald E. Knuth，John L. Hennessy，Larry L. Peterson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、研究及珍藏。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”已经出版了近两百个品种，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。其影印版“经典原版书库”作为姊妹篇也被越来越多实施双语教学的学校所采用。

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。随着计算机科学与技术专业学科建设的不断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将步入一个新的阶段，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。华章分社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：

华章网站：www.hzbook.com
电子邮件：hzjsj@hzbook.com
联系电话：(010) 88379604
联系地址：北京市西城区百万庄南街1号
邮政编码：100037

译 者 序

两位给软件工程和计算机科学专业的学生教授了15年软件测试课程的老师，历时7年，融合大量课堂经验，终于磨砺出这本《软件测试基础》。我同样从事了多年软件测试技术教学，看到这本书时，不禁感慨这是一本不可多得的教科书和参考书。

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，或围绕一个典型的软件开发周期的各个阶段展开，这样的方法使原本的测试主题变得难懂，而本书正是依靠其独特之处成为教科书或参考书的上佳选择。

经过大量的课堂检验，本书对于软件测试采用了可谓独具匠心的理解。它将软件测试定义为把许多定义良好的、通用的测试标准应用到软件结构或模型的过程，同时是生产高质量软件的一个不可或缺的实践工程活动。

本书用一种新颖而简单的结构把测试覆盖标准中复杂、晦涩的观点组织起来。从技术侧面说，软件测试是基于满足覆盖标准的。本书的观点是，真正不同的覆盖标准很少，各种覆盖标准很容易归为以下4类：图形、逻辑表达式、输入空间和语法结构。这不仅简化了测试，而且也易于将每个分类直接地理论化处理。传统的方法将开发过程中各个阶段的测试区别对待，而该方法与之形成鲜明对比。

本书的写作风格直接，从基础讲解概念，把所需的背景知识保持在最低，通篇包含了大量的实例，它把测试当作了客观的、可测量的和重复的量化活动的集合，同时也在必要的地方提出理论概念以支持测试工程师的后续实践活动。

本书采用模块化设计，彼此间相互关联，合理搭配，可以用于多种课程。书中的大部分内容仅需要基础的离散数学和编程知识就可以了。

本书在理论和实践应用之间保持了巧妙的平衡，重点讲解如何管理测试过程和测试者基于基础理论的具体测试技术，并且特别注重设计和创建设计测试用例的基本技术问题，旨在综合软件开发的整个过程，覆盖尽可能多的技术。

本书同时也可以使不同的角色从中受益。对于学生，本书使其可以学到软件测试背后的基本原理，学到如何应用这些原理来更快、更好地生产软件；对于教师，大量的练习、启发式的问题、课堂上的幻灯片和给出的课外活动使得教师很容易教授这些材料；企业的测试者，将发现本书收集了帮助提高他们测试水平的技术。

致谢

译者近几年来一直在北京大学软件与微电子学院从事软件测试技术的研究与授课。在翻译本书的过程中，学院对我的工作给予了极大的支持与重视。我的硕士研究生张瑒、李磊、张坚、伍晓东、赵文博、曹宇奇等同学参与了本书的部分翻译与整理工作，付出了很多努力，在此表示感谢。

郁 莲

20010年6月

前　言

本书把软件测试当作生产高质量软件必不可少的一个工程实践活动。可以作为本科生或研究生软件测试课程的主要教材，也可以作为软件工程或数据结构等一般课程的补充材料，或者作为软件测试工程师和开发人员的资源。这本书有许多独特之处：

- 本书用一种新颖而简单的结构把测试覆盖标准中复杂的、晦涩难懂的观点组织起来。从技术层面上来说，软件测试是以满足覆盖标准为基础的。本书的中心观点是，真正不同的覆盖标准很少，各种覆盖标准很易于归为以下四类：图形、逻辑表达式、输入空间和语法结构。这不仅简化了测试，而且也易于将每个分类直接地加以理论化处理。传统的方法将开发过程中各个阶段的测试区别对待，而该方法与之形成鲜明对比。
- 本书是作为一本教科书来构思和撰写的，写作风格直截了当，从基础讲解概念，把所需的背景知识降到最低。本书包括了大量的例子、家庭作业和教辅材料。它在理论和实际应用之间保持了很好的平衡，把测试当作客观的、可测量和重复的量化活动的集合。本书会在必要的地方提出理论概念以支持测试工程师的后续实践活动。
- 本书认为，测试是帮助IT专业人士开发更高质量软件的一种智力训练。测试不是一个反工程化的活动，也不是一个具有内在破坏性的过程。本书不仅仅是针对测试专家的，也面向对编程或数学知之甚少的领域专家。
- 本书采用模块化结构，彼此间相互关联，所以可以用于多种课程。书中的大部分内容仅需要基础的离散数学和编程知识，需要更多背景知识的地方会清晰地标明。正如在前言的稍后部分描述的那样，对讲述内容进行合理搭配，本书可以用于多种课堂。
- 本书假定，读者学习的目标是成为用最低的成本做出最好的软件的工程师。书中的概念在理论上是非常基础的，但也是非常实用的，大多是当前正在使用的。

为什么要用这本书

不久前，软件开发公司可能会雇用不会测试的程序员和不会编程的测试者。对于大多数企业来说，双方没有必要知道对方背后的技术原理。在行业历史上，软件测试一直被当成一个非技术性的活动。软件企业主要从管理和过程的角度来看测试，对测试者的技能培训没抱多大期望。

随着软件工程行业日趋成熟，软件已渗透到人们日常生活的各个方面，对软件可靠性、可维护性和安全性的要求越来越高。软件企业必须用多种方法来应对这些变化，其中就包括改善软件测试方法。这需要提高测试工程师的专业技能，还需要不断强调软件开发者做测试的重要性。值得庆幸的是，经过30多年的研究和实践，已涌现出许多知识和技术。本书把这些知识整理起来，使学生、测试工程师、测试管理者和开发者都能使用。

同时，我们也发现在大学中教授测试的课程相对较少。只对少数本科生安排了测试课，对计算机科学或软件工程专业的硕士生几乎没有安排软件测试课程，少数十几个教学课程中安排一个测试选修课。测试不仅没有成为本科生计算机科学教育的必要部分，而且大部分计

算机科学的学生根本就没有任何测试知识，或者只是作为一般课程的一部分，在软件工程课上讲几节课。

本书的作者给软件工程和计算机科学专业的学生讲授软件测试已经超过15年了。在此期间，我们得出了一个很不期望看到的结论：没有人写出一本我们想要的书。所以，如果我们想要，就必须自己写。

以前的测试类书籍把软件测试当作一个相对简单的学科，认为这个学科依赖于过程，而不是从技术的角度去理解软件是如何构成的；有的书把测试作为一个需要详细理解大量软件开发技术的复杂的、割裂的学科；还有的书把测试当作一个只有数学家和计算机理论科学家才能掌握的纯理论学科。大多数关于测试的书籍围绕着一个典型的软件开发周期的各个阶段展开，这种方法会使原本普通的测试主题变得难懂。最后，大多数测试类书籍是作为参考书而写的，而不是教科书。所以，只有先前有过专门软件测试知识的教师才能轻松使用。**而本书对于那些不是测试专家的教师也是易于使用的。**

本书在许多重要方面不同于其他的软件测试类书籍。许多书讲解如何管理测试过程，当然这很重要，但告诉测试者基于基础理论的具体测试技术同样重要。本书在理论和实践应用之间保持了很好的平衡。这是软件公司必须有的重要信息，但是本书特别注重设计和创建测试用例的基本技术问题。目前市面上其他的测试类书籍主要关注技术或活动，比如系统测试或单元测试，而本书旨在综合软件开发的整个过程，涵盖尽可能多的技术。

如前所述，本书的目的是支持多种软件测试课程。我们在乔治·梅森大学软件工程硕士研究生的软件测试课上做了第一个尝试，每学期有30多个计算机科学和软件工程专业的学生选修这门课程。我们还组织了软件测试的博士研讨会，举办了特定方向的短期企业培训，还为许多本科课程进行了讲座。虽然有关软件测试的本科课程不多，但我们相信不久的将来会有很多。许多关于测试的书并不是用在课堂上的，我们特地写了这本书来支持课堂教学，因此在本书网站（www.cs.gmu.edu/~offutt/softwaretest）上本书目录的后面看到我们的测试课程的提纲就不足为奇了。

本书采用了许多精心打造的实例来帮助学生和老师学习略显复杂的概念。教辅资源包括高质量的PPT、演讲提示、习题解答和相关软件。我们的思想是：我们不仅仅是在写一本书，同时也在为社区提供课程。我们的目标之一是，所写的内容作为研究文献有学术性，对于非研究人员也是易于使用的。虽然本书的论述与出自研究论文的资料有些不同，但其本质思想是忠实于文献的。为了使文章更为通顺，我们删除了论述中的引用。对于那些喜欢追根溯源的研究者，每章最后都有一个参考文献注释，它对概念的来源进行了总结。

本书的读者对象

学生通过本书可以学到软件测试背后的基本原理，学到如何应用这些原理来更快、更好地生产软件。他们不仅会成为优秀的程序员，而且他们还可以为未来的雇主实施高质量的软件测试活动。教师即使没有软件测试的实际经验，也可以在课堂上使用本书。即使教师不是软件测试方面的专家，本书附带的大量的练习、启发式的问题、课堂上的幻灯片和给出的课外练习也会使他们很容易教授这些课程。研究生（如一年级的博士生）会发现这本书是介入这一领域的无价之宝。清晰合理地讲述理论，用实际应用说明哪些有用，哪些没用；用高级阅读和参考文献把对这些内容感兴趣的读者引向相关资料。虽然软件测试的研究生相对较少，

但是我们认为他们是关键读者群，因为通俗、低门槛会使研究生较容易加入测试研究者社区。已经熟悉这一领域的研究者会发现“标准-方法”新颖而有趣。可能有人不同意这种教学方法，但是我们发现，把测试当作将有限的几个标准应用到极少数的软件结构中的观点对我们的研究很有帮助。我们希望将来的测试研究能够从寻找更多的标准转向对现有标准的创新使用和评测上。

企业中的测试者将发现本书收集了帮助他们提高测试水平的宝贵技术，无论他们当前的水平如何。这里所提出的标准更倾向于成为发现缺陷的技巧“工具箱”。阅读这本书的开发者将发现大量改善软件的方法。他们的自测活动将变得更快速、更有效，关于测试工程师找出软件缺陷的讨论将帮助开发者避开它们。正如一个很有名的寓言所讲的，如果你想教一个人成为好的渔夫，就要讲解鱼如何以及在哪里游泳。最后，管理者将发现本书很好地解释了聪明的测试工程师如何做好他们的工作，以及测试工具如何运行。这样，他们在雇人、晋升和购买工具时，就可以做出更有效的决定。

如何使用本书

本书结构的一个主要优势是，它能够轻松地用于多种不同的课程。大多数教材依赖大学和高中所教授的知识：数据结构和离散数学的基本概念。本书各个部分的组织使得每章前面的内容对于低年级或者基础不太好的学生也很易于使用，需要更高级知识的材料都会明确标出。

特别地，本书定义了7个单独的章节集合，它们组成了贯穿本书的模块分类：

1）计算机科学专业大学二年级课程模块。

2）软件测试专业大学二年级水平的课程模块。

3）一般软件工程课程模块。

4）软件测试高级课程模块。

5）理学硕士一年级水平的软件测试课程模块。

6）面向研究的研究生高级软件测试课程模块。

7）从业人员的相关章节模块。

这种模块分类方法可见后面的简略目录。每个章节都标明了它属于哪个模块。当然，有的教师、学生和读者更愿意按照自己的兴趣或目标来使用这些模块。我们的建议是，第1章的前两节和第6章的前两节用于在数据结构（CS II）课程中阅读，它们后面都有一个简单的作业。我们最喜欢的是让学生们找一个他们以前写的评过分的程序，然后，让程序满足某种简单的测试标准，比如分支覆盖。每发现一个缺陷，我们都给出分数，这使人理解了两个概念：一是“A”级并不意味程序总可以运行，二是发现缺陷才是一件好事情。

	模块						
	1	2	3	4	5	6	7
第一部分　概览							
第1章　概述	■	■	■	■	■	□	□
1.1　测试工程师的工作	■	■	■	■	■	□	□
1.2　软件测试的局限性和术语	■	■	■	■	■	□	□
1.3　测试覆盖标准		■	■	■	■	□	□
1.4　以往的软件测试术语					■	□	
1.5　参考文献注释						□	

（续）

	模块						
	1	2	3	4	5	6	7
第二部分　覆盖标准							
第2章　图覆盖		■	■	■	▭	▭	▭
2.1　概述		■	■	■	▭	▭	▭
2.2　图覆盖标准		■	■	■	▭	▭	▭
2.3　源代码的图覆盖		■	■	■	▭	▭	▭
2.4　设计元素的图覆盖				■	▭	▭	▭
2.5　规格说明的图覆盖				■	▭	▭	▭
2.6　用例的图覆盖				■	▭	▭	▭
2.7　用代数方法表示图					▭	▭	
2.8　参考文献注释						▭	
第3章　逻辑覆盖		■	■	■	▭	▭	▭
3.1　概览：逻辑谓词和子句		■	■	■	▭	▭	▭
3.2　逻辑表达式覆盖标准		■	■	■	▭	▭	▭
3.3　程序的结构化逻辑覆盖		■		■	▭	▭	
3.4　基于规约的逻辑覆盖				■	▭	▭	
3.5　有限状态机的逻辑覆盖			■	■	▭	▭	
3.6　析取范式标准					▭	▭	
3.7　参考文献注释						▭	
第4章　输入空间划分			■	■	▭	▭	▭
4.1　输入域建模			■	■	▭	▭	▭
4.2　组合策略标准			■	■	▭	▭	▭
4.3　划分中的约束					▭	▭	▭
4.4　参考文献注释						▭	
第5章　基于句法的测试		■	■	■	▭	▭	▭
5.1　基于句法的覆盖标准		■	■	■	▭	▭	▭
5.2　基于程序的语法			■	■	▭	▭	▭
5.3　集成与面向对象测试				■	▭	▭	
5.4　基于规范的语法					▭	▭	
5.5　输入空间语法		■	■	■	▭	▭	▭
5.6　参考文献注释						▭	
第三部分　在实践中运用的标准							
第6章　实际的考虑	■	■	■	■	▭	▭	▭
6.1　回归测试	■	■	■	■	▭	▭	▭
6.2　集成和测试	■	■	■	■	▭	▭	▭
6.3　测试过程			■	■	▭	▭	▭
6.4　测试计划			■	■	▭	▭	▭
6.5　识别正确的输出		■	■	■	▭	▭	
6.6　参考文献注释						▭	
第7章　技术的工程标准				■	▭	▭	▭
7.1　测试面向对象软件				■	▭	▭	▭
7.2　测试Web应用和Web服务					▭	▭	▭
7.3　测试图形用户界面					▭	▭	▭
7.4　实时软件和嵌入式软件					▭	▭	▭
7.5　参考文献注释						▭	

（续）

	模块						
	1	2	3	4	5	6	7
第8章 创建测试工具					■	□	□
8.1 图和逻辑表达式标准的插桩					■	□	□
8.2 构造变异测试工具						□	
8.3 参考文献注释						□	
第9章 软件测试中的挑战					■	□	□
9.1 测试紧急性属性：安全性和保密性					■	□	□
9.2 软件的可测试性					■	□	□
9.3 测试标准和软件测试的未来					■	□	
9.4 参考文献注释						□	

大学二年级水平的软件测试课程（模块2）紧跟着数据结构课程。标出部分的材料只需要数据结构和离散数学的知识。

一般的软件工程课程模块（模块3）可以增加这些课程的文献综述材料。标出部分提供了软件测试的基础知识。

软件测试的高级课程（模块4）是本书的主要部分。它增加的材料就软件开发而言需要比大学二年级模块更多的背景知识。这包括第2章中有关数据流测试的章节，涉及多个模块集成测试的章节和依赖于文法或有限状态机的章节。大多数三年级计算机科学专业的学生已经在其他的课程中看到过这些材料。出现在模块4但不在模块2中的大多数章节，可以加上适当的简介到模块2中。值得注意的是，测试工程师使用数据流测试时不需要懂得所有的语法分析理论，要使用测试状态图也不需要知道所有有限状态机的理论。

研究生水平的测试课程（模块5）增加了许多章节，这些章节需要大量的理论上有待成熟的背景知识。例如，某些章节需要初级形式化方法、多态和UML图的知识。许多高级主题和构建测试工具的整章内容也是为研究生准备的，可以为一个好项目提供一个基础，比如，实现一个简单的覆盖分析器。

研究生的高级软件测试课程（模块6）侧重于研究，就像一个博士的研讨会，包括仍未证实或还在研究的问题。参考文献注释是给那些想在将来进一步深入阅读的学生准备的。

最后，在模块7中标注出来的章节可广泛地用于企业中，特别是那些有商业工具支持的企业。这些章节涉及的理论最少，省略了那些在可用性上仍有问题的标准。

在本书的网站上可以获取大量的补充材料，包括教学大纲、PPT、演讲提示、习题解答、可运行的软件和勘误表。

致谢

在撰写本书的过程中，许多人帮助过我。乔治·梅森大学软件测试课上的学生们不仅能够非常宽容地使用本书的半成品，而且还很热情地提供反馈来改进本书。我们不能把这些学生一一列出（有10个学期的学生都使用了本书），下面仅列出了做出突出贡献的几位学生：Aynur Abdurazik、Muhammad Abdulla、Yuquin Ding、Jyothi Chinman、Blaine Donley、Patrick Emery、Brian Geary、Mark Hinkle、Justin Hollingsworth、John King、Yuelan Li、

Xiaojuan Liu、Chris Magrin、Jyothi Reddy、Raimi Rufai、Jeremy Schneider、Bill Shelton、Frank Shukis、Quansheng Xiao和Linzhen Xue。我们特别感激那些给本书提出无私的评论的人：Guillermo Calderon-Meza、Becky Hartley、Gary Kaminski和Andrew J.Offutt。我们感谢其他教育机构的初期使用者提出的反馈：Roger Alexander、Jane Hayes、Ling Liu、Darko Marinov、Arthur Reyes、Michael Shin和Tao Xie。我们也想感谢许多为本书提供素材的人：Roger Alexander、Mats Grindal、Hong Huang、Gary Kaminski、Robert Nilsson、Greg Williams、Wuzhi Xu。我们很高兴收到他们的建议：Lionel Briand、Renée Bryce、Kim King、Sharon Ritchey、Bo Sanden和Steve Schach。很感谢我们的编辑Heather Bergman提供的坚定支持，他为我们的项目推迟了截稿时间，也同样感谢剑桥大学出版社的Kerry Cahill为这个项目提供的强大支持。

我们还要感谢乔治·梅森大学为我们两个人提供公休，还在关键时刻为我们提供助教。我们的系主任Hassan Gomaa也为这个项目提供了热情的支持。

最后，这本书的完成也离不开我们家人的支持。谢谢Becky、Jian、Steffi、Matt、Joyce和Andrew让我们稳定地生活，感谢他们在过去的5年中给我们带来的快乐。

正如所有程序都有缺陷一样，所有的书都有错误。我们的书也是如此。软件缺陷的责任在于开发者，而书中内容错误的责任在于我们作者。值得一提的是，参考文献注释部分反映了我们对于软件测试领域的观点，这项工作庞大而繁琐。我们提前为书中的纰漏致歉，敬请指正。

Paul Ammann
Jeff Offutt

目　录

第三部分 在实践中运用的标准

第一部分 概 览

第1章 概 述

软件测试的思想和技术已经成为所有软件开发者必备的知识。一个软件开发人员在其职业生涯中必会经常用到本书中所提到的概念。本章介绍了软件测试的一些主题内容，包括描述测试工程师的工作，定义一系列的关键术语，并且解释测试覆盖的核心概念。

软件已经遍布我们整个社会，在众多的设备和系统中它已成为关键的组成部分。软件定义了网络路由器行为、金融网络行为、电话交换网行为、因特网以及现代生活的其他基础设施行为。无论是对于像飞机、宇宙飞船、交通管理系统这样特殊的物件，还是对于像手表、烤箱、汽车、DVD播放器、车库自动门、手机以及远程遥控器这些家常的物品，软件都已经成为这些嵌入式应用的必要的组成部分。现代家庭有至少50个处理器，而有些新车则超过了100个处理器。乐观的消费者认为这些运行着的软件永远不会出错！虽然有很多因素影响着可靠软件的制作，这其中当然包括精心的设计和过程管理，但是，测试仍然是业界用于评估正在开发的软件的最首要的方法。幸运的是，对于多种多样的大量的软件应用来说，只用为数不多的基本的软件测试概念就可以进行测试设计了。本书的一个目标就是介绍这些概念，使得学生或是在职的工程师可以轻松地把它们应用于任何的软件测试情况。

本书与其他的软件测试类书是有所区别的，主要体现在如下几个方面。最首要的不同在于它如何看待测试技术。Beizer在他的代表作《软件测试技术》(Software Testing Techniques)中写道，测试很简单，测试人员所要做的只是“找到一个图然后覆盖它”。 得益于Beizer的卓见，对于我们来说很明显，现有的文献中大量的测试技术有很多的共同点，这与我们初次阅读这些文献的感受是不一样的。测试技术一般结合特定软件工件（artifact）的情况（例如需求文档或代码）或是软件生命周期的特殊阶段（例如需求分析或实现）来陈述的。不幸的是，这样的表述模糊了技术之间潜在的相似性，本书则阐明了这些相似性。

事实上图并没有完全体现出所有测试技术的特点，所以，我们还需要一些其他的抽象模型。令我们惊喜的是，我们已经发现有少数抽象模型能够满足我们的需要：图（graph）、逻辑表达式（logical expression）、输入域特征（input domain characterization）和句法描述(syntactic description)。本书最大的贡献在于将覆盖标准分为四大类，以此来简化测试，这也是本书第二部分分为四章的原因所在。

本书将理论与实践应用相结合，从而将测试展示为一系列客观的可度量、可复现的量化行为。该理论基于已出版的文献，没有过多地拘泥于形式。最重要的是，当测试工程师的实践活动需要有理论支持的时候，我们会介绍理论概念。也就是说，本书是供软件开发人员使用的。

1.1　测试工程师的工作

在本书中，测试工程师（test engineer）是专业的信息技术（Information Technology，IT）人员，他负责一到多项技术型测试活动，包括设计测试输入，生成测试用例值，执行测试脚本，分析测试结果，以及向开发人员和经理报告测试结果。虽然我们以测试工程师为角色进行描述，但是参与软件开发的每个工程师都应该意识到，在某些时候自己扮演着测试工程师的角色。原因在于在产品开发过程中所产生的软件工件都有（或应该有）相应的测试用例集，而最适合定义这些测试用例的通常是工件的设计者。一个测试经理负责一到多个测试工程师。测试经理制定测试策略和过程，在项目上与其他经理相互合作沟通，另外还会帮助测试工程师们工作。

图1.1说明了测试工程师的一些主要活动。测试工程师必须创建测试需求，以此来设计测试用例。这些需求之后会转化成用于测试执行的实际值和脚本。这些可执行的测试是在软件上运行的，在图中用P表示，而对测试结果的评估决定了这些测试是否揭露出软件的错误。这些活动是由1人或多人完成的，而测试的整个过程由测试经理进行监控。

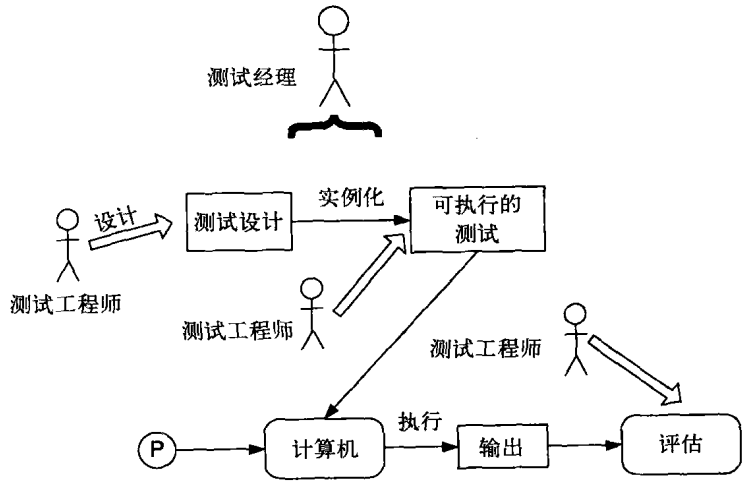


图1.1　测试工程师的活动

测试工程师最有力的工具之一就是正式的覆盖标准。正式的覆盖标准给测试工程师提供方法，来决定在测试中要用哪些测试输入，使测试人员更有可能发现程序中的问题，并且能够对软件的高质量和高可靠性提供更强有力的保障。覆盖标准也为测试工程师提供了测试停止准则。本书的技术核心是介绍当前可用的覆盖标准，描述工具（商用或其他）是如何支持这些覆盖标准的，解释如何最好地应用它们，并且提出如何将它们集成到整个开发流程中。

长久以来，软件测试活动按级别分类，其中有两种级别是经久沿用的。最常用的级别分类是基于传统的软件过程步骤。虽然大多数的测试类型只有在一部分软件实现了以后才能执行，但是在软件开发各步骤都可以进行测试的设计和构造。测试中最花费时间的事实上是测试的设计和构造，因此测试活动可以也应该贯穿开发的整个过程。第二级别分类是基于测试人员的态度和看法的。

1.1.1 基于软件活动的测试级别

测试可以由需求和规约、设计工件或源代码派生。不同的测试级别，伴随着不同的软件开发活动：

- 验收测试：根据需求评估软件；
- 系统测试：根据体系结构设计评估软件；
- 集成测试：根据子系统设计评估软件；
- 模块测试：根据详细设计评估软件；
- 单元测试：根据代码实现评估软件。

图1.2展示了各个测试级别的一个典型场景，以及这些测试级别与分步的软件开发活动是如何分别对应关联的。每个测试级别中的信息都典型地由相应关联的开发活动所派生出来。的确，标准的建议是在每个开发活动的同时就把相应的测试用例设计好，虽然在实现阶段以前软件还是不可执行的。这个建议的理由是仅仅明确、清晰地说明测试的流程可以识别设计决策的缺陷，否则这些设计决策看上去似乎是合理的。迄今为止，早期发现缺陷是减少最终开支的最有效方法。应该注意到，这幅图并不是就意味着它是瀑布式开发过程。综合与分析这些活动可以广泛地应用于任何开发过程。

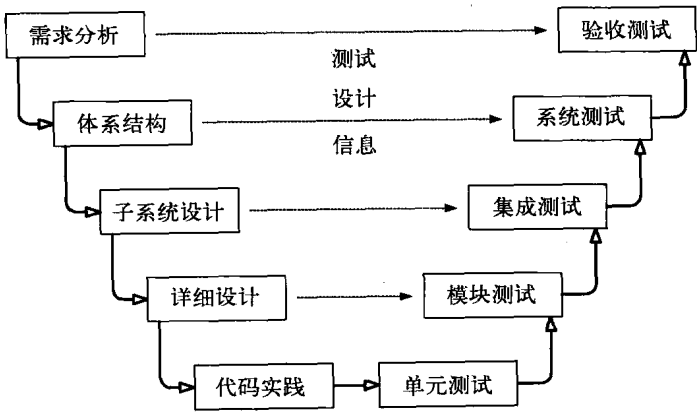


图1.2 软件开发活动与测试级别——“V模型”

软件开发过程的*需求分析*阶段是为了捕捉用户的需求。设计*验收测试*是为了确定软件成品事实上是否满足了这些需求。换言之，验收测试是为了查明软件是否是用户所需要的。验收测试必须有用户或那些拥有很好的领域背景知识的其他人员参加。

软件开发过程的*体系结构*设计阶段，是选择组件（component）和连接器（connector），二者结合来实现一个系统以便符合先前所确定的需求。设计*系统测试*是为了确定集成后的完整的系统是否与规约一致。系统测试假设各个部分单独工作正常，而且要问系统作为一个整体是否工作正常。这一级别的测试通常是为了寻找设计和规约中存在的问题。在该阶段，寻找更低级别的错误代价是昂贵的，而且通常不是由程序员来做而是由独立的测试组来完成。

软件开发过程的*子系统*设计阶段详细说明子系统的结构和行为，每个子系统分别实现整个体系结构中的某个功能。通常，子系统采用之前已开发好的软件。设计*集成测试*是为了评估子系统内模块（在下面定义）之间接口是否一致、通信是否正确。集成测试的前提是各个模块能正确工作。有些测试文献中把术语集成测试与系统测试互换使用；在本书中，集成测

试不涉及测试被集成的系统或者子系统。集成测试通常是由开发组成员负责的。

软件开发过程的详细设计阶段确定各个模块的结构和行为。一个程序单元（unit）或过程（procedure），是由一个或多个连续的程序语句组成，并有一个名字，软件其他部分使用该名字对程序单元进行调用。在C和C++中，单元称为函数（function）；在Ada中称为过程或函数；在Java中称为方法（method）；在Fortran中称为子例程（subroutine）。将一系列相关联的单元组合在一个文件、包或者类中，就称为一个模块（module）。相当于C中的文件，Ada中的包，C++和Java中的类。设计模块测试是为了独立地评估各个模块，包括组件单元间如何相互作用，以及关联的数据结构。大多数软件开发组织将模块测试的职责交给程序员。

软件开发过程的实现阶段是实际产生代码的阶段。设计单元测试是为了评定实现阶段所产生的单元的正确性，它是最底层的测试。有些时候，比如在建一些通用的库模块时，单元测试无需封装软件应用的知识。跟模块测试一样，大部分软件开发组织会把单元测试的责任交给程序员。直截了当的做法是用一些像测试Java类的JUnit的工具将单元测试与相关的代码一起打包。

在图1.2中没有描述回归测试（regression testing），它是软件开发维护阶段的一个标准部分。回归测试是在对软件进行一些修改之后进行的测试，目的在于帮助确保更新后的软件仍然具备更新前所拥有的功能。

在需求和高级别设计中存在的错误都会导致最终实现的程序中的缺陷；测试可以揭露出这些缺陷。不幸的是，由于需求和设计中的错误而导致的缺陷，在这些原始的错误之后数月或数年内只有通过测试才能展现出来。这些错误的影响往往会蔓延到多个软件构件中，因此这样的错误通常很难加以约束并且修正的代价很高。从积极的一面看，即使不进行测试，定义测试用例的过程本身就已经可以鉴别出一些重大的需求和设计错误。因此，在需求分析和设计的同时就并行地推行测试计划是很重要的，而不要将测试计划工作推迟到项目后期。幸运的是，在标准的软件实践中，用一些像用例分析（use-case analysis）这样的技术，测试计划能更好地与需求分析相结合。

虽然很多文献中强调测试所应用的级别，但事实上更重要的区分是在于我们所要找的**缺陷类型**。这些缺陷是基于我们所测试的软件**工件**，以及我们生成测试用例所用到的软件**工件**。例如，单元测试和模块测试是分别基于单元和模块的，而我们通常试图发现在独立地执行单元和模块测试时所发现的错误。

区分单元测试与系统测试的一个最好的例子是声名狼藉的奔腾错误。在1994年，英特尔公司推出奔腾微处理器，而几个月之后，弗吉尼亚州Lynchburg大学的一名数学家Thomas Nicely就发现对于某些浮点除法运算，芯片会给出错误的运算结果。

芯片对于某些数字的组合会有微小的不精确性，英特尔声称（可能是真的）只有90亿分之一的除法运算会出现精确度降低的问题。这个错误是由于除法算法所用到的一个含有1 066个数值的表中有五个数据遗漏。这五个输入地址应该包含常数+2，而事实上这五个输入地址没有初始化而用0代替了。麻省理工学院的数学家Edelman称“奔腾的这个错误是个很容易犯的错误，但是很难捕捉”，一个有关遗漏某个基本点的分析。在系统测试中这是很难被发现的，而事实上，英特尔声称用这个表进行了数百万次的测试。但是表中这五个输入地址仍被留空，这是由于一个循环的退出条件没有写对，即在循环完成之前就停止了存储数据的操作。这在单元测试中其实是很容易发现的，分析显示几乎任何单元级覆盖标准都能找到这个数百万美