HZ BOOKS
华章教育

计 算 机 科 学 丛 书

CRC Press
Taylor & Francis Group

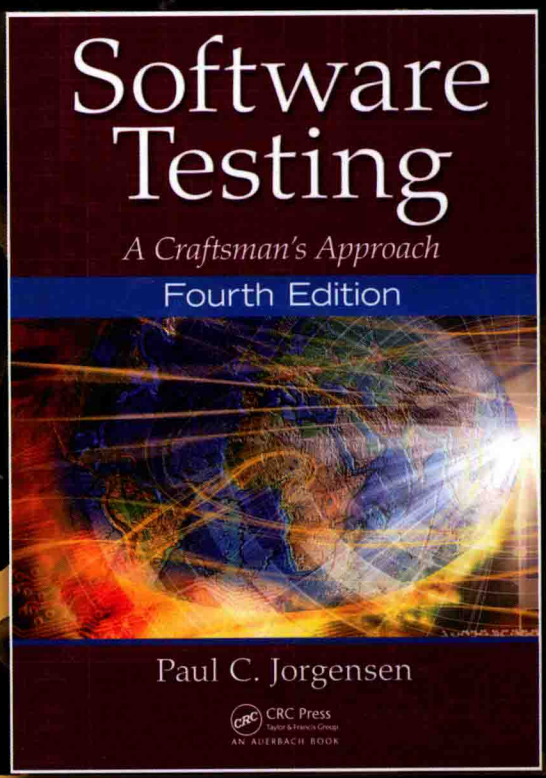
原书第4版

软件测试

一个软件工艺师的方法

[美] 保罗 C. 乔根森（Paul C. Jorgensen）著
马琳 李海峰 译

Software Testing
A Craftsman's Approach Fourth Edition



机械工业出版社
China Machine Press

计 算 机 科 学 丛 书

原书第4版

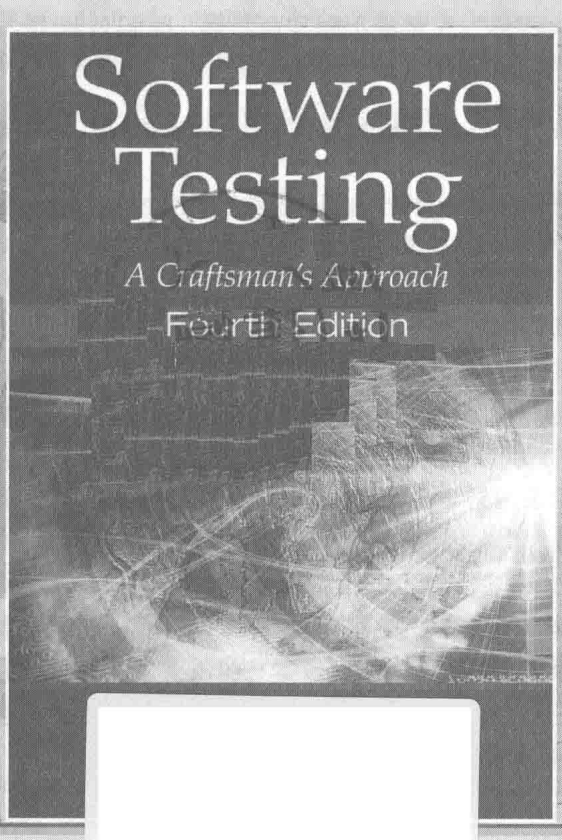
软件测试

一个软件工艺师的方法

[美] 保罗 C. 乔根森（Paul C. Jorgensen）著
马琳 李海峰 译

Software Testing

A Craftsman's Approach Fourth Edition



机械工业出版社
China Machine Press

图书在版编目（CIP）数据

软件测试：一个软件工艺师的方法（原书第 4 版）/（美）保罗 C. 乔根森（Paul C. Jorgensen）著；马琳，李海峰译 . —北京：机械工业出版社，2017.10
（计算机科学丛书）
书名原文：Software Testing: A Craftsman's Approach, Fourth Edition

ISBN 978-7-111-58131-4

I. 软… II. ① 保… ② 马… ③ 李… III. 软件工具 - 测试 - 教材 IV. TP311.562

中国版本图书馆 CIP 数据核字（2017）第 246261 号

本书版权登记号：图字 01-2014-3571

本书是经典的软件测试教材。书中对基础知识、方法提供了系统的综合阐述，既涉及基于模型的开发，又介绍了测试驱动的开发，做到了理论与实践的完美结合，反映了软件标准和开发的最新进展和变化。

出版发行：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：陈佳媛　　责任校对：李秋荣
印　　刷：三河市宏图印务有限公司　　版　　次：2017 年 11 月第 1 版第 1 次印刷
开　　本：185mm × 260mm 1/16　　印　　张：21
书　　号：ISBN 978-7-111-58131-4　　定　　价：79.00 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　　投稿热线：（010）88379604
购书热线：（010）68326294 88379649 68995259　　读者信箱：hzjsj@hzbook.com

文艺复兴以来，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；也正是这样的优势，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公司较早意识到“出版要为教育服务”。自 1998 年开始，我们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，Brian 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相助，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，有的还专门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：

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

华章教育
华章科技图书出版中心

译 者 序

Software Testing: A Craftsman's Approach, Fourth Edition

人总是会犯错的。在软件的开发过程中，有些错误是显式的，它们可以由编译器发现，而更多的错误却是难以发现甚至难以重现的。如何确保软件产品在实际工作中“不出错”，是我们时刻都要面对的一个非常现实的问题。然而，如何在茫茫代码中找到这些错误？如何评价软件测试的结果？测试后的软件是否还残留有缺陷？残留的缺陷对软件有什么影响？这些问题都缺少一个统一的答案，更多的是依靠个人经验。因此，我们认为，软件开发人员应该是软件测试的行家，软件测试人员也应该是软件开发的高手。将这些个人经验整理成有指导意义的资料并共享，这对于软件测试来说显得尤为重要。

原著作者 Paul C. Jorgensen 一直在数学系和计算机系授课，同时还有 20 多年的工业软件开发和管理经验，这些经历使得他能够很好地理解软件测试。本书就是他融合近年来在软件测试教学工作中的心得体会编写而成的。在第 3 版的译者序中，我们曾这样形容这本书：“对软件测试理论与技术介绍得层次分明、全面精到；以若干实例为线索展开内容、循序渐进，便于读者掌握；在很多章节的最后，通过深入的对比和讨论，深刻地阐述和总结了在软件测试中普遍存在的实际问题，精辟深刻；此外，原书在语言上还经常不拘一格，多有诙谐灵动之处。”随着第 4 版的发布，我们看到了作者更为深刻的认识和更为流畅的表达。这一版重新规划了篇章结构，同时根据美国相关标准的要求，扩充了路径测试章节中复杂条件测试和修正的条件判定覆盖率指标等内容，另外，这一版新增了许多紧跟时代、有实用价值的内容。新增的“软件技术评审”章节（第 22 章）是基于一个软件开发公司 20 多年来的实际经验撰写的；“软件复杂度”章节（第 16 章）增加了对面向对象编程和系统层面测试的复杂度的处理；在“测试用例的评估”章节（第 21 章）中，则增加了对愈发普及的变异测试的介绍，用以同漏洞挖掘和故障注入方法进行对比。

本书第 1 ～ 7 章和附录由李海峰翻译，第 8 ～ 23 章由马琳翻译，马琳对全书进行了统稿。翻译在某种程度上是再创作的过程，我们在忠实于原文的同时，针对软件测试工作所包含的各种理论与技术的细节进行了深入的整理和推敲。对基础理论和技术概念尽可能采用相关学科的主流说法，对新技术和新名词尽可能使用当前业界流行的说法，以期更标准、更科学和更易于理解地表达原文。对举例时讲述的故事，则力求通俗流畅，以保持原书的语言风格。

好的作品必然是经过了反复修改才日臻完善的，译著也是一样。由于时间有限，我们无法对所有细节都反复推敲，加上知识水平和实际工作经验有限，不当之处在所难免，恳请读者和同行批评指正。

译者

2017 年 7 月于哈工大

此次再版，我们增加了四章新内容，同时更加深入地讨论了基于路径的测试，从而拓展了本书 18 年以来一直侧重基于模型测试的传统。此前本书已经再版三次，经过了 18 年的教学和业界使用的检验。借助精心挑选的简单易懂的实例，本书把理论与实践紧密地结合在一起。此外，很多第 3 版中的内容被合并、重组在一起，使全书内容更加简洁流畅。把很多面向对象软件测试的内容和过程软件测试（procedural software testing）整合在一起形成了一个有机的整体。还有就是针对美国联邦航空管理局和美国国防部有关标准的要求，在“路径测试”一章中扩充了复杂条件测试（complex condition testing）和修正的条件判定覆盖率（modified condition decision coverage）指标等内容。

这一版新增加的章节如下：

- 软件技术评审（第 22 章）。侧重软件技术检验，这实际上被视为“静态测试”，而本书的前三版一直侧重于讨论如何利用精心挑选的测试用例来执行代码的“动态测试”。本章内容实际上来自一个软件开发公司 20 多年来的产业实践经验，该公司具有完善的技术评审流程。
- 附录。附录中给出了一套完整的用例集（采用 UML），可以针对典型的客户需求实现实际产业开发所要求的技术检验。其中包括用例标准、用例故障严重程度定义、潜在问题的技术检验事项表，以及典型评审报告和最终报告的格式文档。
- 基于模型的综合系统测试（第 17 章）。由系统构成复杂系统的问题相对还是较新的（始于 1999 年）。软件测试从业人员现在是在追随几位大学研究人员的步伐，主要关注如何界定一个由若干系统构成的复杂系统。这一章介绍了“泳道事件驱动 Petri 网”，在表达能力上接近著名的状态图方法。有了它就可以对复杂系统实施基于模型的测试。
- 软件复杂度（第 16 章）。目前大部分文献都仅考察了在单元层面上的圈复杂度（cyclomatic，也称为 McCabe）。本章从两个方面拓展了对单元层面复杂度的考量，引进了两种集成层面上的复杂度。对面向对象编程和系统层面的测试来说，需要涉及对复杂度的处理。在任何层面上，对复杂度的考量都是提升设计、编码、测试和维护工作的重要手段。保持一种一致的软件复杂度表述，对每个阶段都有很大的促进作用。
- 测试用例的评估（第 21 章）。新增的这一章要研究一个难题：如何评估一个测试用例集？测试覆盖性是长期以来为人所接受的指标，但是其中总有一定程度的不确定性。古罗马关于“谁来守卫卫兵”的问题，在此变成了“谁来评估测试”的问题。十几年来，变异测试（mutation testing）逐渐成为一种解决方案，所以本章对其效果和贡献进行了介绍，同时也介绍了另外两种方法：漏洞挖掘（fuzzing）和故障注入（fault insertion）。

做了47年的软件开发人员和大学教授，我认为自己的软件测试知识既有深度也有广度。在大学里，我一直在数学系和计算机系授课，同时我还有20多年的工业软件开发和管理经验，这些经历使我能够很好地编写和改进我的软件测试教程并不断加深对于软件测试的理解。在我讲到书本以外的内容时，我经常会不断地产生新的看法。所以，我把本书的出版视为我对软件测试领域做出的一点贡献。最后，我还要感谢我的三位同事Roger Ferguson博士、Jagadeesh Nandigam博士和Christian Trefftz博士，感谢他们在面向对象测试这几章的撰写中给予我的巨大帮助。

非常感谢！

Paul C. Jorgensen

于密歇根州罗克福德市

第一部分

Software Testing: A Craftsman's Approach, Fourth Edition

数学基础

第 1 章

Software Testing: A Craftsman's Approach, Fourth Edition

测试概述

为什么要测试？最主要的目的有两个：一是对质量或可接受性做出评判，二是发现存在的问题。之所以要测试，是因为人经常会出错，特别是在软件领域和采用软件控制的系统中，这个问题尤为突出。本章给出软件测试的总体知识框架。

1.1 基本概念

在许多软件测试方面的文献中，名词术语的使用都比较混乱（有时候前后不统一），究其原因，可能是因为测试技术在近几十年中不断地演化进步，而且文献作者的造诣也各有千秋。国际软件测试认证委员会（ISTQB）提出了一个全面的测试术语列表（见 http://www.istqb.org/downloads/glossary.html）。全书所采用的术语同 ISTQB 术语表一致，并且也符合电子电气工程师学会（IEEE）计算机协会 1983 年所颁布的技术标准（IEEE，1983）。这里我们首先来研究几个有用的术语。

- **错误**（error）：人是会出错的。错误的同义词是过失（mistake）。编程时出的错我们称之为 bug。错误很容易传递和放大，比如需求分析时出的错在系统设计时有可能会被放大，而且在编码时还会被进一步放大。
- **故障**（fault）：故障是错误的后果。更确切地说，故障是错误的具体表现形式，比如文字叙述、统一建模语言（UML）图表、层次结构图、源代码等。类似于把编程错误称为 bug，故障的一个很好的同义词是缺陷（defect）（见 ISTQB 术语表）。故障可能难以捕获。比如一个遗漏错误所导致的故障可能只是在表象上丢掉了一些应有的内容。这里给我们的启发是有必要把故障进一步细分为过失故障和遗漏故障。如果在表象中添加了不正确的信息，这是过失故障；而未输入正确的信息，则是遗漏故障。在这两类故障中，遗漏故障更难检测和纠正。
- **失效**（failure）：代码执行时发生故障就会导致失效。失效具有两个很微妙的特征：1）失效只出现在程序的可执行表象中，通常是源代码，确切地说是加载后的目标代码；2）这样定义的失效只和过失故障有关。那么如何处理遗漏故障所对应的失效呢？进一步说，对于不轻易被执行的故障，或者长期不执行的故障，情况又会怎样呢？实施代码审查（见第 22 章）能够找出故障来避免失效。实际上，好的代码审查同样能检查出遗漏故障。
- **事故**（incident）：当失效发生时，用户（或客户，测试人员）可能明显察觉到，也可能察觉不到。事故是与失效相关联的症状，它警示用户有失效发生了。
- **测试**（test）：测试显然要考虑到错误、故障、失效和事故等诸多问题。测试就是利用测试用例来试验软件。测试有两个明确的目标：找出失效和证实软件执行的正确性。
- **测试用例**（test case）：每个测试用例都有一个用例标识，并针对一项程序行为。每个测试用例还包括一组输入和期望输出。

图 1-1 给出了一个软件测试的生命周期模型。从图中可以看出，在软件开发阶段，有三

个地方可能会产生错误，由此引发的故障将会传递到后续开发过程中。故障解决阶段实际上也是一处可能产生错误（以及新故障）的地方。如果实施了一个修复操作会导致原先正确的软件出现异常行为，这样的修复就是不完善的。本书后面在讨论回归测试时还要进一步研究这个问题。

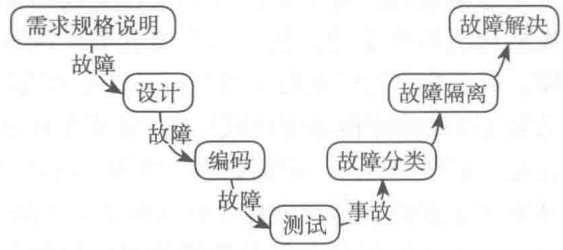


图 1-1　软件测试的生命周期

从这一系列术语可以看出，测试用例在测试中占核心地位。测试的过程还可以进一步细分为若干独立的步骤：测试计划制订、测试用例开发、测试用例运行，以及测试结果评估等。本书的重点就是研究如何构造有效的测试用例集合。

1.2　测试用例

软件测试的精髓是为被测对象找到一组测试用例。测试用例是（或者应该是）被承认的工作产品。一个完整的测试用例包括测试用例标识符、简短的目的描述（例如一个业务规则）、前置条件描述、实际的测试用例输入、期望输出、期望的后置条件描述和执行记录。执行记录主要用于测试管理，可以包括执行测试的日期、执行人、针对的软件版本，以及测试是否通过。

测试用例的输出部分常常会被忽视。这是很不应该的，因为输出通常是测试用例最难的部分。比如我们假设你正在测试一个软件，针对美国联邦航空管理局（FAA）的航线限制和当天的气象数据，这个软件要给飞机确定最佳航线。然而你又怎么知道这个最佳航线到底是什么呢？可能会有各种各样的答案。从学术角度来看，任何问题都一定会有一个确切的答案。从行业角度来看，可以采用“参考测试”（reference testing）的办法，由专家用户来参与系统的测试。对于给定的一组测试用例输入，由这些专家来评判系统的输出是否可以接受。

运行测试用例包括建立必要的前置条件，给出测试用例输入，观察输出结果，将实际输出与期望输出进行比较，然后在保证预期后置条件成立的情况下，判断测试能否通过。由此可以看出，测试用例显然是非常有价值的，至少和源代码一样珍贵。所以，也需要对测试用例进行开发、审查、使用、管理和保存。

1.3　利用维恩图来理解软件测试

从本质上讲，测试关心的是软件的行为，而软件行为同软件（或系统）开发人员所常用的面向程序代码的视角并没有直接关系。此处最明显的差别在于：代码侧重于“软件是什么”，而行为则关注“软件干什么”。一直困扰测试人员的一个难点问题是：基础性的文档通常都是由开发人员编写，并且为开发人员服务的，所以这些文档就很自然地强调程序代码方面的信息而不是软件行为信息。本节将给出一个简单的维恩图来说明软件测试中的一些很微妙的问题。

我们先来看看程序的行为空间（注意，此处我们专注研究的是测试的本质）。对于给定的程序及其规格说明，考察其规格说明所规定的行为集合 S 和编程实现的行为集合 P。图 1-2 给出了规格行为和实现行为之间的关系。在程序的所有可能行为中，规定行为都在圆圈 S 内，所有实际实现的行为都在圆圈 P 中。利用这个维恩图，可以清楚地看到测试人员所

遇到的问题。如果某些规定行为未经编程实现，情况会怎样呢？用前面提到的术语来讲，这些是遗漏故障。类似地，如果编程实现的某些行为不是规格说明所规定的，情况又会怎样呢？这些就是过失故障，或者在满足了规格说明之后又发生的错误。S 与 P 的交集（图中的橄榄球型区域）是“正确”的部分，即那些按规定实现的行为。对测试的一个很好的认识是：测试就是要确定按规定实现的程序行为到底有多少。（需要补充的是，正确性仅仅是针对确定的规格说明和具体的程序实现而言的。正确性是相对的，而不是绝对的。）

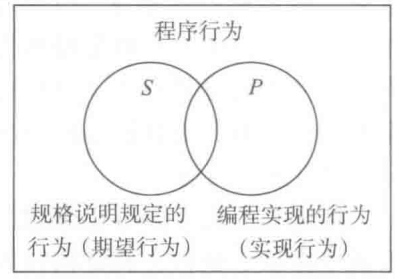


图 1-2　程序的规定行为与实现行为

在图 1-3 中，新圆圈代表测试用例集合。注意，它同程序行为全空间以及各个行为集合之间存在些许不同。一个测试用例要引发一个程序行为，这里请数学家们谅解这种表述中的不严密性。所以考察集合 S、P 和 T 之间的关系可以看到：可能会存在测试不到的规定行为（2 号区和 5 号区），测试到的规定行为（1 号区和 4 号区），以及对应于未规定行为的测试用例（3 号区和 7 号区）。

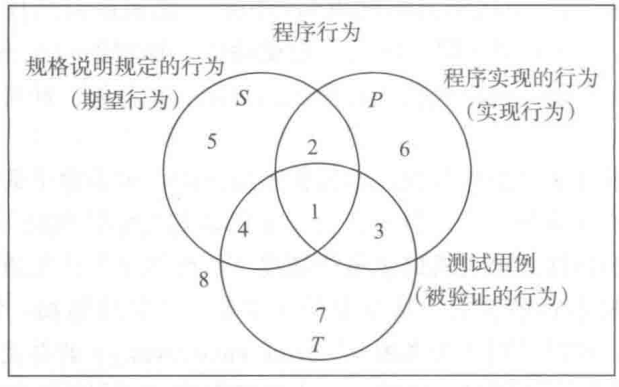


图 1-3　规定行为、实现行为和测试行为

类似地，也会有测试不到的实现行为（2 号区和 6 号区），测试得到的实现行为（1 号区和 3 号区），以及对应于未实现行为的测试用例（4 号区和 7 号区）。

在维恩图中，每一个区域都很重要。如果某些规定行为没有相应的测试用例，那测试就是不完备的。如果有测试用例对应的是未规定行为，则会产生几种可能：此测试用例设计得不恰当，或者规格说明不够充分，又或者测试人员故意要确认规定不该发生的行为确实不会发生。（就我个人的经验来讲，好的测试人员经常会有意设计最后一种情况的测试用例。这也是在进行规格说明和系统设计评审时吸收有经验的测试人员参与的重要原因。）

至此我们已经可以看清把软件测试称为技艺或工艺的原因了：测试人员怎样才能尽可能地扩大所有行为集合的交集（1 号区）呢？也就是说，应该如何确定集合 T 中的测试用例呢？最简单的回答就是：遵循测试方法来构造测试用例。这个思路给了我们一种方法来比较各种测试方法的有效性，详见第 10 章。

1.4　构造测试用例

构造测试用例有两种基本方法，传统上称为功能测试和结构测试。但用基于规格说明的测试和基于代码的测试则能表述得更确切些，所以本书将采用这两个名称。每种方法都有几