HZ BOOKS
华章科技

软 件 工 程 技 术 丛 书

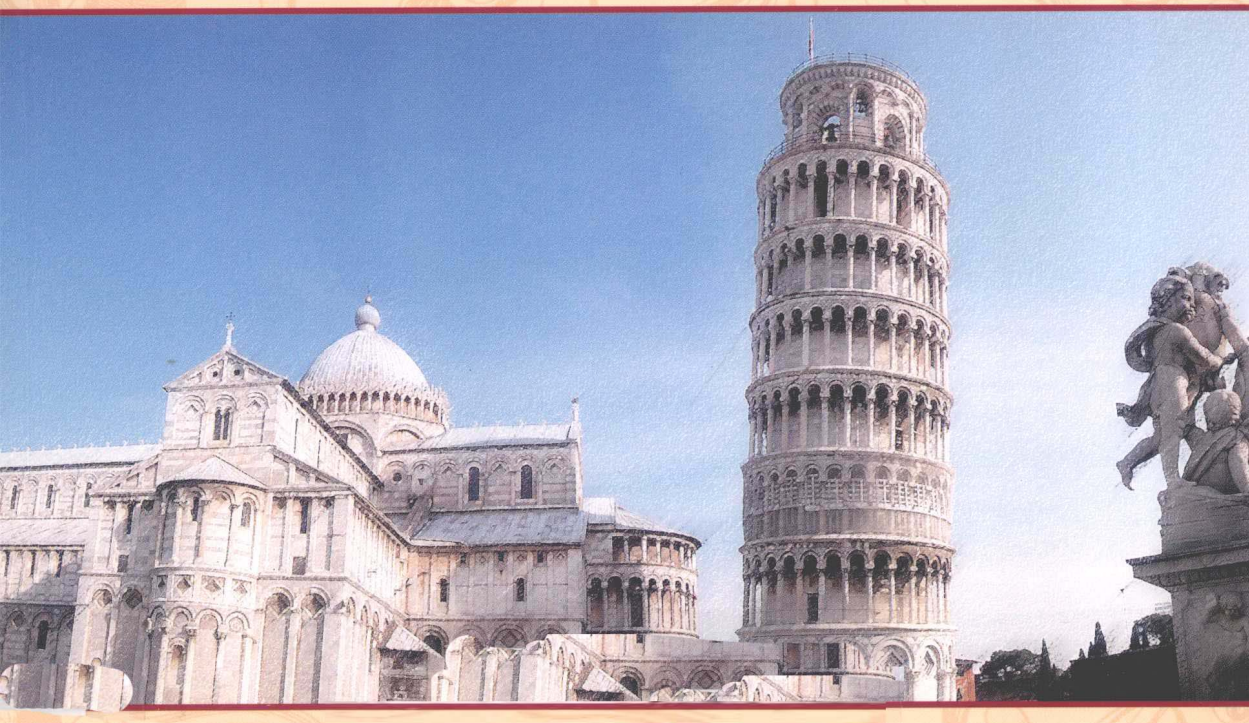
WILEY

软件测试的艺术

（原书第3版）

The Art of Software Testing (Third Edition)

（美）Glenford J. Myers Tom Badgett Corey Sandler 著 张晓明 黄琳 译



机械工业出版社
China Machine Press

软件工程技术丛书

软件测试的艺术

（原书第3版）

The Art of Software Testing (Third Edition)

（美）Glenford J. Myers Tom Badgett Corey Sandler 著 张晓明 黄琳 译

机械工业出版社
China Machine Press

本书从第1版付梓到现在已经30余年，是软件测试领域的经典著作。本书结构清晰、讲解生动活泼，简明扼要地展示了久经考验的软件测试方法和智慧。

本书以一次自评价测试开篇，从软件测试的心理学和经济学入手，探讨了代码检查、走查与评审、测试用例的设计、模块（单元）测试、系统测试、调试等主题，以及极限测试、互联网应用测试等高级主题，全面展现了作者的软件测试思想。第3版在前两版的基础上，结合软件测试的最新发展进行了更新，覆盖了可用性测试、移动应用测试以及敏捷开发测试等内容。

本书适合软件开发人员、IT项目经理等相关读者阅读，还可以作为高等院校计算机相关专业软件测试课程的教材或参考书。

Glenford J. Myers, Tom Badgett, Corey Sandler: The Art of Software Testing, Third Edition (ISBN: 978-1-118-03196-4).

本书版权登记号：图字：01-2011-7872

图书在版编目（CIP）数据

软件测试的艺术（原书第3版）/（美）梅耶（Myers, G. J.）等著；张晓明，黄琳译. —北京：机械工业出版社，2012.3
（软件工程技术丛书）
书名原文：The Art of Software Testing, Third Edition

ISBN 978-7-111-37660-6

Ⅰ. 软…　Ⅱ. ①梅…　②张…　③黄…　Ⅲ. 软件测试　Ⅳ. TP311.5

中国版本图书馆CIP数据核字（2012）第040000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：吴　怡
北京市荣盛彩色印刷有限公司印刷
2012年4月第1版第1次印刷
170mm × 242mm · 12.75印张
标准书号：ISBN 978-7-111-37660-6
定价：39.00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

译者序

《软件测试的艺术》（下简称《艺术》）作为元老级的测试书在国内可能没那么出名，但它的确非常经典且很有口碑，书中所提出的“软件测试为求错而非求证”的观点至今仍在学术界被广泛争议与讨论。随着软件测试的重要性越来越受到现代软件企业的重视，这本书就好像尘封已久的宝藏被人们挖掘出来并受到追捧。与此同时，也正是因为测试市场的需求激增，书店里的测试书籍也似乎是“忽如一夜春风来，千树万树梨花开”了。

我和《艺术》的第一次接触并不是在书店里发现了它，而是因为我阅读的一个习惯。我喜欢在阅读之前先看参考文献部分，也由此发现国内的很多软件测试书籍都把《艺术》作为首要的参考书目，这让我不得不对该书刮目相看，现在我终于明白了，原来《艺术》一书就是现在各种测试书籍参考的源头之一。以我个人的观点，今天书店里的软件测试理论书籍（注意我是指理论方面）已经饱和甚至是富营养化，如果你打算系统地学习软件测试理论知识，我不敢向你保证这本书是最全面最详细的，但是绝对是恰到好处的，它精悍凝练的篇幅可以让你在最短时间内获得关于软件测试的真知灼见。

对于那些已经成为测试工程师甚至是高级测试工程师的人来说，本书同样值得一读，书中的很多内容读起来仿佛醍醐灌顶，本书所涵盖的测试知识经过千锤百炼和时间的考验，而把这些理论知识结合你的测试经验，能系统化并巩固加深你对测试这门学科的理解，而这种对软件测试技术系统的、深刻的理解，将使你在今后的工作以及事业中受益匪浅。

因此，作为一名测试工程师，我对读者的建议是，初学者可将本书作为入门书；而有经验者更应该将本书作为理论指南，花点时间翻阅一下，梳理自己的经验和知识；本书对开发人员也相当有用，可以让你在最短的时间内建立起对测试的框架认知，从而在编码的过程中能够在脑海里多一些测试的思想，十分有益，当然有些测试类型本身就需要开发者参与，比如本书所介绍的极限编程与测试，开发者需

要编写单元测试用例。对于测试管理者而言，本书的重要性不言而喻，本书内容非常精炼，将有助于你根据项目情况制定更合理、更有效的测试计划。

作为本书第3版的译者，我十分有幸能够接到这样的经典书籍翻译任务，而且还是本人的处女译，心中的忐忑自不必多说，只希望我的翻译能够不辱原著的经典，更能够得到读者的认可。

此次第3版相对于之前的第2版增加了全新的两章（第7章和第11章），由此使得本书包含了当下最新的测试分类和技术。除了第9章的改动增补较大外，其他的章节基本是略有改动。

在翻译过程中，我尽量保证在准确翻译原文的基础上增加了一些个人的见解，通常以译者注的形式给出，我相信这些注释有助于国内的读者理解和消化书中内容；也有一些地方做了勘误。有时候直译的效果并不那么好，我也会尝试一些意译，比如在11.4节，作者强调了移动应用必然越来越火这一现象和趋势，我便借用著名诗歌《见与不见》来意译：

你用，或者不用

移动应用就在那里

……

因为是站在巨人的肩膀上（前两版译文）进行翻译，所以我必须感谢前两版的翻译人员，这使得我此次翻译的精力主要聚焦在新增的两章以及改动较大的章节上。同时要感谢黄琳为我的原始译稿所做的详尽审稿和勘误工作，这也是我们之间的二度合作。

最后请大家注意书名中的“艺术”二字，这暗示本书并不是那么晦涩难懂或写得很深奥，我觉得即使你没多少计算机基础知识，只要用过电脑软件，本书的很多章节读起来应该都不会太吃力。我希望有越来越多各行各业的人们加入测试的世界，从本书第7章的可用性测试我们看到，测试需要领域内经验，如果某公司针对音乐爱好者推出了一款混音软件，他们肯定会想，要是能够把周杰伦请来做首席用户体验师该有多好。总之，测试，这是个相比较开发来讲门槛不算太高的职业（当然要做到精深未必容易，甚至难度还要高），而且收入还算不错，在这里与各位读者共勉了。

张晓明

2012年2月6日

序　言

1979年，Glenford J. Myers出版了一本现在仍被证明为经典的著作，这就是本书第1版。本书经受住了时间的考验，25年来一直列在出版商提供的书目清单中。这个事实本身就是对本书可靠、精粹和珍贵品质的佐证。

在同一时期，本书第3版的几位合著者共出版了200余本著作，大多数都是关于计算机软件的。其中有一些很畅销，再版了多次（例如Corey Sandler的《Fix Your Own PC》自付梓以来已出版到第8版，Tom Badgett关于微软PowerPoint及其他Office组件的著作已经出版到第4版）。然而，那些作者的著作中没有哪一本书能够像本书一样持续数年之后仍畅销不衰。

区别究竟在哪里呢？那些新书只涵盖了短期性的主题：操作系统、应用软件、安全性、通信技术及硬件配置。20世纪80年代和90年代以来的计算机硬件与软件技术的飞速发展，必然使得这些主题频繁变动和更新。

在此期间出版的有关软件测试的书籍已数以百计，这些书也对软件测试的主题进行了简要的探讨。然而，本书为计算机界一个最为重要的主题提供了长期、基本的指南：如何确保所开发的所有软件做了其应该做的，并且同样重要的是，未做其不应该做的？

本书第3版中保留了同样的基本思想。我们更新了其中的例子以包含更为现代的编程语言。我们还研究了在Myers编著本书第1版时尚无人了解的主题：Web编程、电子商务、极限编程与测试及移动应用测试。

但是，我们永远不会忘记，新的版本必须遵从其原著，因此，新版本依然向读者展示Glenford Myers全部的软件测试思想，这个思想体系以及过程将适用于当今乃至未来的软件和硬件平台。我们也希望本书能够顺应时代，适用于当今的软件设计人员和开发人员掌握最新的软件测试思想及技术。

前　言

在本书1979年第1版出版的时候，有一条著名的经验，即在一个典型的编程项目中，软件测试或系统测试大约占用50%的项目时间和超过50%的总成本。

30多年后的今天，同样的经验仍然成立。现在出现了新的开发系统、具有内置工具的语言以及习惯于快速开发大量软件的程序员。但是，在任何软件开发项目中，测试依然扮演着重要角色。

在这些事实面前，读者可能会以为软件测试发展到现在不断完善，已经成为一门精确的学科。然而实际情况并非如此。事实上，与软件开发的任何其他方面相比，人们对软件测试仍然知之甚少。而且，软件测试并非热门课题，本书首次出版时是这样，遗憾的是，今天仍然如此。现在有很多关于软件测试的书籍和论文，这意味着，至少与本书首次出版时相比，人们对软件测试这个主题有了更多的了解。但是，测试依然是软件开发中的“黑色艺术”。

这就有了更充足的理由来修订这本关于软件测试艺术的书，同时我们还有其他一些动机。在不同的时期，我们都听到一些教授和助教说：“我们的学生毕业后进入了计算机界，却丝毫不了解软件测试的基本知识，而且在课堂上向学生介绍如何测试或调试其程序时，我们也很少有建议可提供。”

因此，本书再版的目的与前两版一样：填充专业程序员和计算机科学学生的知识空缺。正如书名所蕴涵的，本书是对测试主题的实践探讨，而不是理论研究，还包括对新的语言和过程的探讨。尽管可以根据理论的脉络来讨论软件测试，但本书旨在成为实用且“脚踏实地”的手册。因此，很多与软件测试有关的主题，如程序正确性的数学证明都被有意地排除在外了。

第1章介绍了一个供自我评价的测试，每位读者在继续阅读之前都须进行测试。它揭示出我们必须了解的有关软件测试的最为重要的实用信息，即一系列心理和经济学问题，这些问题在第2章中进行了详细讨论。第3章探讨的是不依赖计算机的代码走查或代码检查的重要概念。不同于大多数研究都将注意力集中在概念的

过程和管理方面，第3章则是从技术上“如何发现错误”的角度来进行探讨。

读者可能会意识到，在软件测试人员的技巧中最为重要的部分是掌握如何编写有效测试用例的知识，这正是第4章的主题。第5章探讨了如何测试单个模块或子例程，第6章讲述了如何测试更大的对象。第7章围绕用户体验或可用性测试这一重要的软件测试概念进行阐述，在更复杂且拥有更广大用户量的软件不断涌现的今天，可用性测试变得越来越重要。第8章介绍了一些程序调试的实用建议，第9章着重研究了极限编程及其测试（一种在今天称为敏捷开发环境中的编程和测试方法）。第10章介绍如何将本书所涵盖的软件测试知识运用到Web开发中，包括电子商务系统以及社交网络[⊖]的开发。第11章描述了如何测试移动设备上的应用。

本书主要面向三类读者。第一类是专业的程序员。尽管我们希望本书的内容对于他们来说不是全新的知识，但本书能使专业程序员对测试技术增强了解。如果这些材料能使软件开发人员在某个程序中多发现了一个错误，那么本书创造的价值将远远超过书价本身。

第二类读者是项目经理，他们将会直接受益于本书所介绍的实用的测试管理理论与知识。第三类读者是软件或计算机专业的学生，我们的目的在于向学生展示程序测试的问题，并提供一系列有效的技术。对于最后一类读者群，我们建议本书作为程序设计课程的补充教材，使学生在学习阶段的早期就接触到软件测试的内容。

Glenford J. Myers

Tom Badgett

Todd M. Thomas

Corey Sandler

⊖ 具有高度的和用户交互性质的Web应用，也即是所谓的Web 2.0。——译者注

目　录

第 1 章
Chapter 1

一次自评价测试

自本书30年前首次出版以来，软件测试变得比以前容易得多，也困难得多。

软件测试何以变得更困难？原因在于大量的编程语言、操作系统以及硬件平台的涌现。在20世纪70年代只有相当少的人使用计算机，而在今天几乎人人离不开计算机。而今天计算机不仅仅是指摆在你书桌上的计算机了，几乎所有我们所接触和使用的电子设备都内置了一个“计算机”或者计算芯片，以及运行在其上的软件系统。不妨回想一下在今天的社会中还在使用哪些不需要软件驱动的设备，没错，锤子和手推车是，但是这些工具也大量使用在由软件控制和操作的车间中。软件的普遍应用提升了测试的意义。今天的设备已经千百倍强于它们的“前辈”，今天的“计算机”这个概念也变得越来越广泛和越来越难准确地定义。数字电视、电话、游戏产品、汽车等都有一颗计算机的“心”以及运行其中的软件，以至于在某些情况下它们自己本身也能够被看做是一台特别的计算机。

因此，现在的软件会潜在地影响到数以百万计的人，使他们更高效地完成工作，反之也会给他们带来数不清的麻烦，导致工作或事业的损失。这并不是说今天的软件比本书第1版发行时更重要，但可以肯定地说，今天的计算机（以及驱动它的软件）无疑已影响到了更多的人、更多的行业。

就某些方面而言，软件测试变得更容易了，因为大量的软件和操作系统比以往更加复杂，内部提供了很多已充分测试过的例程供应用程序集成，无须程序员从头进行设计。例如，图形用户界面（GUI）可以从开发语言的类库中建立起来，同时，由于它们是经过充分调试和测试的预编程对象，将其作为自定义应用程序的组成部分进行测试的要求就减少了许多。

另外，尽管市场上的测试书籍越来越多，甚至有过剩之嫌，似乎依旧有很多开发人员对全面的测试并不那么欢迎。引入更优秀的开发工具、使用已经通过测试

的GUI（图形界面控件）控件、紧张的交付日期以及高度集成的便利开发环境会让测试变得仅仅是让那些最基本的测试用例走走过场罢了。影响不大的bug也许只不过会让最终用户觉得使用不方便而已，然而严重的bug则可能造成经济损失甚至是人身伤害。本书所阐述的方法旨在帮助设计人员、开发工程师以及项目经理更好地理解全面综合测试的意义所在，并提供行之有效的指南以帮助达成测试的目标。

所谓软件测试，就是一个过程或一系列过程，用来确认计算机代码完成了其应该完成的功能，不执行其不该有的操作。软件应当是可预测且稳定的，不会给用户带来意外惊奇。在本书中，我们将讨论多种方法来达到这个目标。

好了，在开始阅读本书之前，我们想让读者做一个小测验。我们要求设计一组测试用例（特定的数据集合），适当地测试一个相当简单的程序。为此要为该程序建立一组测试数据，程序须对数据进行正确处理以证明自身的成功。下面是对该程序的描述：

> 这个程序从一个输入对话框中读取三个整数值，这三个整数值代表了三角形三条边的长度。程序显示提示信息，指出该三角形是何种三角形：不规则三角形、等腰三角形还是等边三角形。

注意，所谓不规则三角形是指三角形中任意两条边不相等，等腰三角形是指有两条边相等，而等边三角形则是指三条边相等。另外，等腰三角形等边的对角也相等（即任意三角形等边的对角也相等），等边三角形的所有内角都相等。

用你的测试用例集回答下列问题，借以对其进行评价。对每个回答“是”的答案，可以得1分：

1. 是否有这样的测试用例，代表了一个有效的不规则三角形？（注意，如1、2、3和2、5、10这样的测试用例并不能确保“是”的答案，因为具备这样边长的三角形不存在。）
2. 是否有这样的测试用例，代表一个有效的等边三角形？
3. 是否有这样的测试用例，代表一个有效的等腰三角形？（注意，如2、2、4的测试用例无效，因为这不是一个有效的三角形。）
4. 是否至少有三个这样的测试用例，代表有效的等腰三角形，从而可以测试到两等边的所有三种可能情况（如3、3、4；3、4、3；4、3、3）？
5. 是否有这样的测试用例，某边的长度等于0？

6. 是否有这样的测试用例，某边的长度为负数？

7. 是否有这样的测试用例，三个整数皆大于0，其中两个整数之和等于第三个？（也就是说，如果程序判断1、2、3表示一个不规则三角形，它可能就包含一个缺陷。）

8. 是否至少有三个第7类的测试用例，列举了一边等于另外两边之和的全部可能情况（如1、2、3；1、3、2；3、1、2）？

9. 是否有这样的测试用例，三个整数皆大于0，其中两个整数之和小于第三个整数（如1、2、4；12、15、30）？

10. 是否至少有三个第9类的测试用例，列举了一边大于另外两边之和的全部可能情况（如1、2、4；1、4、2；4、1、2）？

11. 是否有这样的测试用例，三边长度皆为0（0，0，0）？

12. 是否至少有一个这样的测试用例，输入的边长为非整数值（如2.5、3.5、5.5）？

13. 是否至少有一个这样的测试用例，输入的边长个数不对（如仅输入了两个而不是三个整数）？

14. 对于每一个测试用例，除了定义输入值之外，是否定义了程序针对该输入值的预期输出值？

当然，测试用例集即使满足了上述条件，也不能确保能查找出所有可能的错误。但是，由于问题1至问题13代表了该程序不同版本中已经实际出现的错误，对该程序进行的充分测试至少应该能够暴露这些错误。

开始关注自己的得分之前，请考虑以下情况：以我们的经验来看，高水平的专业程序员平均得分仅7.8（满分14）。如果读者的得分更高，那么祝贺你。如果没有那么高，我们将尽力帮助你。

这个测验说明，即使测试这样一个小的程序，也不是件容易的事。因此，想象一下测试一个十万行代码的空中交通管制系统、一个编译器，甚至一个普通的工资管理程序的难度。随着面向对象编程语言（如Java、C++）的出现，测试也变得更加困难。举例来说，为测试这些语言开发出来的应用程序，测试用例必须要找出与对象实例或内存管理有关的错误。

从上面这个例子来看，完全地测试一个复杂的、实际运行的程序似乎是不太可能的。情况并非如此！尽管充分测试的难度令人望而生畏，但这是软件开发中一项非常必需的任务，也是可以实现的一部分工作，通过本书我们可以认识到这一点。

第 2 章
Chapter 2

软件测试的心理学和经济学

软件测试是一项技术性工作，但同时也涉及经济学和人类心理学的一些重要因素。

在理想情况下，我们会测试程序的所有可能执行情况，而在大多数情况下，这几乎是不可能的。即使一个看起来非常简单的程序，其可能的输入与输出组合可达到数百种甚至数千种，对所有的可能情况都设计测试用例是不切合实际的。对一个复杂的应用程序进行完全的测试，将耗费大量的时间和人力资源，这样在经济上是不可行的。

另外，要成功地测试一个软件应用程序，测试人员也需要有正确的态度（也许用“愿景”（vision）这个词会更好一些）。在某些情况下，测试人员的态度可能比实际的测试过程本身还要重要。因此，在深入探讨软件测试的本质之前（指技术层面），我们先探讨一下软件测试的心理学和经济学问题。

2.1 软件测试的心理学

测试执行得差，其中一个主要原因在于大多数的程序员一开始就把“测试”这个术语的定义搞错了。他们可能会认为：

“软件测试就是证明软件不存在错误的过程。”

“软件测试的目的在于证明软件能够正确完成其预定的功能。”

“软件测试就是建立一个‘软件做了其应该做的’信心的过程。”

这些定义都是本末倒置的。

每当测试一个程序时，应当想到要为程序增加一些价值。通过测试来增加程序的价值，是指测试提高了程序的可靠性或质量。提高了程序的可靠性，是指找出并最终修改了程序的错误。

因此，不要只是为了证明程序能够正确运行而去测试程序；相反，应该一开始就假设程序中隐藏着错误（这种假设对于几乎所有的程序都成立），然后测试程序，发现尽可能多的错误。

那么，对于测试，更为合适的定义应该是：

“测试是为发现错误而执行程序的过程”。

虽然这看起来像是个微妙的文字游戏，但确实有重要的区别。理解软件测试的真正定义，会对成功地进行软件测试有很大的影响。

人类行为总是倾向于具有高度目标性，确立一个正确的目标有着重要的心理学影响。如果我们的目的是证明程序中不存在错误，那就会在潜意识中倾向于实现这个目标；也就是说，我们会倾向于选择可能较少导致程序失效的测试数据。另一方面，如果我们的目标在于证明程序中存在错误，我们设计的测试数据就有可能更多地发现问题。与前一种方法相比，后一种方法会更多地增加程序的价值。

这种对软件测试的定义，包含着无穷的内蕴，其中的很多都蕴涵在本书各处。举例来说，它暗示了软件测试是一个破坏性的过程，甚至是一个“施虐”的过程，这就说明为什么大多数人都觉得它困难。这种定义可能是违反我们愿望的；所幸的是，我们大多数人总是对生活充满建设性而不是破坏性的愿景。大多数人都本能地倾向于创造事物，而不是将事物破坏。这个定义还暗示了对于一个特定的程序，应该如何设计测试用例（测试数据）、哪些人应该而哪些人又不应该执行测试。

为增进对软件测试正确定义的理解，另一条途径是分析一下对“成功的”和“不成功的”这两个词的使用。当项目经理在归纳测试用例的结果时，尤其会用到这两个词。大多数的项目经理将没发现错误的测试用例称为一次“成功的测试”，而将发现了某个新错误的测试称为“不成功的测试”。

这又是一次本末倒置。“不成功的”表示事情不遂人意或令人失望。我们认为，如果在测试某段程序时发现了错误，而且这些错误是可以修复的，就将这次合理设计并得到有效执行的测试称做是“成功的”。如果本次测试可以最终确定再无其他可查出的错误，同样也被称做是“成功的”。所谓“不成功的”测试，仅指未能适当地对程序进行检查，在大多数情况下，未能找出错误的测试被认为是“不成功的”，这是因为认为软件中不包含错误的观点基本上是不切实际的。

能发现新错误的测试用例不太可能被认为是“不成功的”，也就是说，能发现错误就证明它是值得设计的。“不成功的”测试用例，会看到程序输出正确的结果