周芝英　郑人杰　译

计算机软件测试技巧

清华大学出版社

计算机软件测试技巧

〔美〕Glenford J. Myers

周之英
郑人杰 译

清华大学出版社

内容简介

程序测试是软件开发方面必不可少的重要工作。本书不是从理论上而是从实际的角度去论述这一课题，是一本实用的手册性读物。第一章为简短的自我测验。第二章详细叙述了测试的一些哲理性问题和经济性的问题。第三章讨论了不使用计算机的测试方法，包括代码人工运行和代码审查会。第四章讨论如何写出有效的测试步骤。五六两章讨论了单个模块或子程序的测试和较大的程序的测试。第七章介绍程序纠错的一些实用方法。第八章综述了测试工具、有待研究的一些领域及其他技术问题。并列出大量参考书目。

本书可供大专院校学生、教师及需学用计算机的科技人员参考。也是开设软件开发方法有关课程的参考书。

The Art of software Testing

GLenford J.Myers

John Wiley & Sons, Inc.New York, 1979

计算机软件测试技巧

周之英 郑人杰译

★

清华大学出版社出版

北京 清华园

河北省香河县印刷厂排版

河北省固安县印刷厂印刷

新华书店北京发行所发行·各地新华书店经售

★

开本：787×1092 1/32 印张：6$\frac{3}{4}$ 字数：145千字

1985年7月第一版 1985年7月第一次印刷

印数：00001～60000

书号：15235·163 定价：1.50元

序　言

人们早已知道，典型的程序设计项目几乎要把一半的时间和一半以上的费用花在测试所开发的程序或系统上。谈到这一点，也许有人会认为，现在程序测试已经发展成为一门“真正的科学”了。其实，目前人们对于程序测试的了解，比起软件开发的其它方面都少。而且，测试一直是一个“不时兴”的课题，或是说，在这方面发表的文章很少。

以上这一点在很大程度上鼓励我以《软件测试技巧》为题写出本书。除此以外，教授和助教们常常告诉我，“我们的学生毕业后，进入了产业部门，但他们还并不懂得如何测试程序。而在学校所学的课程中基本上没有告诉学生究竟应该如何测试他们的程序，如何排除程序中的错误”。

因此，本书要为专业程序员和计算机专业的学生补充这方面知识。正如标题所表明的那样，这本书不是从理论上而是从实际的角度去论述这一课题。虽然我们也可以从理论上讨论程序测试问题，不过本书力图写成一本实用的手册性读物。因此，对于许多与程序测试有关的理论问题，如程序正确性的数学证明都有意略去了。

第一章是一个简短的自我测验，每个读者在读本书时都应先做这个测验。第二章详细叙述了测试的一些哲理性问题和经济性问题，事实证明，这个问题是程序测试最重要的实际问题。第三章讨论了不使用计算机的测试方法，包括代码人工运行和代码审查会两个重要概念。一般讨论这两点时着

眼于过程和管理，本书则从技术上如何发现错误的角度进行讨论。

有经验的读者会懂得，一个程序测试员所掌握的"锦囊"中，最重要的技巧是如何写出有效的测试情况。这是第四章的中心议题。第五章、第六章分别讨论单个模块或子程序的测试，和较大的程序的测试。而第七章则介绍了程序纠错的一些实用方法。第八章综述了测试工具、有待研究的一些领域以及在本书其他章节未讨论的技术问题。此外，还列出大量的参考书目。

本书主要针对三类对象。对专业程序员来说，不一定本书的所有内容都是新的，但我希望读了本书会增加他们测试技术方面的知识，即使本书只帮助他们在某个程序中多查出一个错，也足以补偿本书的代价了。书中所提供的测试过程管理方面新的实际资料，对于设计项目管理人员是有用的。第三类对象是程序设计和计算机科学的学生。本书可以使他们了解程序测试中的问题，并提供一些有效的程序测试技巧。本书亦可作为程序设计课的补充教材，使学生在初学阶段就了解软件测试问题。

Glenford J.Myers

1978年7月于纽约市

目　录

第一章　自我测验

（A Self-Assessment Test）

在开始阅读本书主要内容之前，我们恳切地建议大家做下面的简短测验。问题是测试以下的程序，该程序要完成的工作是：

从卡片上读入三个整数值。这三个数值表示三角形三条边的长度。然后，打印出信息，以表明这个三角形是等腰三角形或是全等三角形，或是一般三角形。

在一张纸上为这个程序写下一套你认为合适的测试情况（即：一些具体的数据组）。做完之后，再往下看，分析你的测验结果。

下一步是评价你的测验结果。这样作了以后就会发现，写这个程序要比最初看上去难得多。由于我们已经对该程序的不同编写方式进行了研究，并且编出一个常见错误表，于是就可以根据你的那套测试情况回答下面的问题，每回答一个“是”给你自己加一分，最后对你那测试情况作出评分。

1. 你是否有一种测试情况表示合法的不规则三角形（注意，象1，2，3和2，5，10这样的测试情况不应回答“是”。因为不存在这样边长的三角形）？
2. 你是否有一种测试情况表示合法的等边三角形？
3. 你是否有一种测试情况表示合法的等腰三角形（2，2，4这样的测试情况不算）？

4. 你是否至少有三种测试情况表示合法等腰三角形，由此检查了两条边相等的所有三种排列方案（如：3，3，4；3，4，3和4，3，3）？

5. 你是否有这样的测试情况，其中三角形的一边长为0？

6. 你是否有这样的测试情况，其中三角形的一条边长为负数？

7. 你是否有一种测试情况，其中三个整数都大于0，而其中的两个数之和等于第三个数（即，如果程序把1，2，3当成了一个不规则三角形则程序有错）？

8. 你是否至少有三种第7类那样的测试情况，检查一条边边长等于另外两边边长之和的所有三种排列方案（如：1，2，3；1，3，2和3，1，2）？

9. 你是否有一种测试情况，表示三个整数都大于0，其中某两个数的和小于第三个数(如1，2，4或12，15，30)？

10. 你是否至少有三种第9类那样的测试情况，检查了所有三种排列的方案（如1，2，4；1，4，2和4，1，2）？

11. 你是否有一种测试情况表示三条边边长都为0（即：0，0，0）？

12. 你是否至少有一种测试情况，给出的边长不是整数？

13. 你是否至少有一种测试情况，给出了错误的数值个数（例如：给出两个整数，而不是三个）？

14. 对于每一种测试情况，除了输入值外，你是否还给出了预期输出？

当然，满足上面条件的一组测试情况不能保证查出所有

可能的错误，但由于问题1—13代表了该程序不同的编写实际上可能发生的错误，所以对程序进行充分的测试应能检查出这些错误。如果你的测验做得很差，说明你有一定的代表性。而经验相当丰富的专业程序设计员平均只得7.8分(满分14分)。这个成绩表明，即使象上面这样简单的程序的测试工作也不是一件容易的事。如果真是这样，请设想一下，测试一个具有十万条语句的空中交通管制系统，一个编译程序，甚至一个普通工资发放程序的困难吧！

第二章　程序测试的心理学问题和经济学问题

(The Psychology and Economics of Program Testing)

尽管人们可以从各种不同的技术观点来讨论软件测试，然而软件测试中最为重要的却是经济学问题和人们的心理学问题。换句话说，如果我们考虑了是否可能对一个程序进行“彻底的”测试，知道了应由谁来完成测试工作，搞清了在测试程序时应该采取的正确态度等等，那我们就比仅仅从纯技术的角度去考虑问题，更有可能取得测试的成功。所以，在我们讨论技术性较强的问题之前，先从这些问题着手是完全合理的。

在考虑心理学和经济学的问题时，我们将用一些篇幅来谈谈程序测试最重要的一件事。一旦解决了这个问题，其它有关程序测试的讨论，都不过好象是蛋糕上的那层糖霜一样成为附属性的东西了。

“测试”这个词的定义似乎很简单，然而却是个至关重要的问题。我们之所以要讨论这个问题，是因为大多数人使用着完全错误的定义，而这正是程序测试效果不好的主要原因。象“程序测试是证明程序中不存在错误的过程”，“程序测试的目的是要证明程序正确地执行了预期的功能”，“程序测试的过程，是使人们确信程序可完成预期要完成的工作的过程”，等等，都是错误定义的例子。

由于这些定义几乎与软件测试的目的完全背道而驰，因而是错误的。让我们暂且放下这些定义，设想一下测试程序的状况。人们在测试程序时自然是想给程序增添一点价值（就是说：程序测试是一项花费昂贵的活动，测试者必然希望通过增加程序自身的价值来补偿一些花费）。而提高程序的价值，就意味着提高程序的质量或可靠性。提高程序的可靠性就意味着发现并改正程序中的错误。所以，进行测试，不应是为了显示程序是好的，而应从程序中含有错误这个假定出发，去测试程序，从中发现尽可能多的错误。因此，较为恰当的定义应该是：

程序测试是为了发现错误而执行程序的过程

也许有人觉得我们的讨论好象只是一场语义学的玄妙游戏，然而人们已经发现测试定义对程序测试的成功具有深远的影响。我们知道，人类的活动具有高度的目的性，建立适当的目标具有重要的心理作用。如果我们的目的是要证明程序中没有错误，那我们就会不自觉地朝这个方向去做；也就是说，我们会倾向于挑选那些使程序出错的可能性较小的测试数据。另一方面，如果我们的目标是要证明程序中有错，那就会选择一些易于发现程序所含错误的测试数据。而后一种态度会比前者给程序增添更多的价值。

这样的程序测试定义有很多推论，其中许多推论分散在全书的各个章节。例如，测试的定义意味着程序测试的过程是具有破坏性的，其程度甚至达到了不可容忍的地步。社会上大多数人的人生观是建设性的，而不是破坏性的。人们倾向于创造一个物品，而不是轻易毁坏一个物品。因此，程序

测试的破坏性的定义使人们对程序测试工作望而生畏。程序测试定义还隐含着如何设计测试情况（测试数据），以及应该由谁和不应由谁来测试一个给定程序等等观点。

强调测试正确定义的另一途径，是分析“成功”和“失败”这两个词的用法，特别是工程管理人员用“成功”和“失败”两词对测试结果进行判别时，更需要对测试有个正确的理解。大多数工程理管人员把没发现错误的测试称为“成功的测试运行”，而常常把查出了新错的测试称之为“失败的”测试。这种情况往往说明他们使用了错误的测试定义，因为“成功的”这个词表示达到了某种目的，而“失败的”则表示令人失望或事不如愿。然而，由于查不出错误的测试浪费了大量时间和金钱，因此使用“成功的”这个修饰词就显得不够恰当了（细心的读者会意识到，由于测试之前，我们不能绝对肯定，说查不出错误。因此我们只能在事后万无一失地提出这种意见。后面我们还会讨论到这一问题）。

同样，决不能把查出了新错误的测试看成失败的测试，恰恰相反，事实证明，这个是有价值的工作。所以，强调测试正确定义的另一办法，是把这两个词的习惯用法颠倒过来。于是，我们说可发现错误的测试称为成功的测试，而使程序产生正确结果的测试称为失败的测试。

让我们考虑一个类似的情况，有一个人感到全身不舒服，去找医生看病。如果医生做了一些检查，却不能诊断出他得了什么病，那我们决不会说这些检查是“成功的”；相反，这些检查是失败的。因为病人白白花了40美元的检查费，他的病却没治好。病人就会怀疑医生的诊断能力了。然而，如果

检查确定了病人患有胃溃疡，那检查就成功了。这时医生可以对症治疗了。所以，看来医学界正确地使用着这些词（显然可类比，当我们开始测试一个程序时，我们应把它当做一个病人）。

所谓这种“程序测试是证明程序中不存在错误的过程”定义的另一问题是：实际上对所有的程序，甚至连最简单的程序，都不可能达到这个目标（如果你对这句话感到难于理解，请现在先相信它，因为我们在本章的后面还会讨论到它）。心理学研究还告诉我们，当人在干一件已经知道是不合适的或不可能做到的事时，往往做得不好。例如：如果让一个人在15分钟解出一个刊登在星期日《纽约时报》上的交叉填字字谜，10分钟后我们会看到这人几乎没一点进展，因为他会感到实际上不可能做到而放弃自己的努力。然而，如果我们要求花4小时解出这题，那也许就会看到他在开头的10分钟内有较大的进展了。把程序测试定义为在程序中找出错误的过程，就使测试成了可以做到的任务，从而克服了心理上存在的问题。

第三个问题与“程序测试的过程是演示程序完成预期要求的过程”。这种习惯定义有关，就是说即使程序完成了预期要求，仍可能含有错误。也就是说，如果程序不按要求工作，它显然有错，但是如果程序做了不要它做的事，它也有错。请考虑第一章中有关三角形的程序。即使我们可以肯定这一程序能正确地区分一般三角形，等腰三角形和等边三角形，但如果程序做了不要它做的工作（也就是：如果程序肯定1，2，3表示一个任意三角形，或者0，0，0表示一个等边三角形），程序依然发生了错误，如果我们把程序测

试看成发现错误的过程，而不是把它作证明程序完成预期要求的过程，则更有可能发现后一类错误。

现在总结一下以上的重要讨论，我们把程序测试看成是设法从程序中查出错误(假定它们是存在的)的破坏性过程是较为恰当的。成功的测试是力图让程序执行失败，而使查错取得进展的过程。当然，人们最终总想通过测试来对程序能否完成预期要求，并且不做不要它做的事建立某种程度的信心，可是达到这一目的的最好方法，是花力气查错。设想有人对你宣称："我的程序是完美的"（没有错误）。对这一声明树立信心的最好方法，是设法驳倒它。设法提出它的不足，而不是去核实在某个输入数据的集合上，程序能正确无误地运行。

§2.1 程序测试的经济学
(The Economics of Testing)

给程序测试定义之后，下一步最好是确定能否通过测试发现所有的错误。我们即将看到即使对非常简单的程序答案也是否定的。一般来说，要查出程序中所有的错误既不现实，也常常是不可能的。下面我们将要看到，这一基本问题涉及到程序测试的经济学问题、测试者对程序所作的假定以及设计测试情况的方法。

2.1.1 黑盒测试(Black—Box Testing)

讨论测试问题的方法之一是研究所谓"黑盒"（或数据驱动、输入／输出驱动）的测试方案。**在应用这种方案时，测试者把程序看成一个黑盒。即完全不考虑程序内部结构和**

内部特性。相反，测试者仅仅关心于寻找使程序未按规范运行的情况，并且仅仅按程序的规范导出测试数据（也就是说，不利用有关程序内部结构的信息）。

如果有人指望以这种方法查出程序中所有的错误，办法只能是使用穷举输入测试。所谓穷举输入测试，就是把所有可能的输入都作为测试情况使用。一定要这样做才能查出所有错误的理由是，例如，即使有人对三角形程序测试了三种等边三角形条件，也绝不能保证程序能正确地检验所有的等边三角形。很可能当程序用特定值3842,3842，3842进行检查时，程序表明这个三角形是一般三角形。由于把程序看成了黑盒，所以发现这种问题的唯一途径，只能是测试每一种输入情况。

要穷举地测试这个三角形程序，就得把所有合法三角形（直到最大整数所表示的三角形）统统变成测试情况。这些测试情况本身就是天文数字，**然而这样做也还并非是穷举的**；仍然不会查出象-3，4，5是个一般三角形，2，A，2是等腰三角形等等这类错误。为了确保查出所有这类错误，人们不仅要测试所有合法的输入，而且还要对所有可能的输入进行测试。因此，要穷举地测试这个三角形程序，实际上就得给出无穷多个测试情况。

如果这样做已经使人感到头痛，那较大程序的穷举测试就更成问题了（如果读者允许我谈论“比无穷大还大的量”的话）。请设想一下对COBOL编译程序这一黑盒进行穷举测试的情况吧。人们不仅要编出代表所有合法COBOL程序（同样，实际上是个无穷的量）的测试情况，而且还得编出所有不合法的COBOL程序的测试情况，以确保编译程序能