（续）

接口错误	其他检查
5. 传递给被调用模块的实参属性是否与其形参属性匹配？	5. 是否遗漏了某个功能？
6. 传递给被调用模块的实参量纲是否与其形参量纲匹配？	
7. 调用内部函数的实参的数量、属性、顺序是否正确？	
8. 是否引用了与当前入口点无关的形参？	
9. 是否改变了某个原本仅为输入值的形参？	
10. 全局变量的定义在模块间是否一致？	
11. 常数是否以实参形式传递过？	

3.4 代码走查

代码走查与代码检查很相似，都是以小组为单位进行代码阅读，是一系列规程和错误检查技术的集合。代码走查的过程与代码检查大体相同，但是规程稍微有所不同，采用的错误检查技术也不一样。

就像代码检查一样，代码走查也是采用持续一至两个小时的不间断会议的形式。代码走查小组由三至五人组成，其中一个人扮演类似代码检查过程中“协调人”的角色，一个人担任秘书（负责记录所有查出的错误）的角色，还有一个人担任测试人员。关于这三到五个人的组成结构，有各种各样的建议。当然，程序员应该是其中之一。我们建议另外的参与者应该包括：

- 一位极富经验的程序员；
- 一位程序设计语言专家；
- 一位程序员新手（可以给出新颖、不带偏见的观点）；
- 最终维护程序的人员；
- 一位来自其他不同项目的人员；
- 一位来自该软件编程小组的程序员。

开始的过程与代码检查相同：参与者在走查会议的前几天得到材料，这样可以专心钻研程序。然而走查会议的规程则不相同。不同于仅阅读程序或使用错误检查列表，代码走查的参与者“使用了计算机”。被指定为测试人员的那个人会带着一些书面的测试用例（程序或模块具有代表性的输入集及预期的输出集）来参加会议。在会议期间，每个测试用例都在人们脑中进行推演。也就是说，把测试

数据沿程序的逻辑结构走一遍。程序的状态（如变量的值）记录在纸张或白板上以供监视。

当然，这些测试用例必须结构简单、数量较少，因为人脑执行程序的速度比计算机执行程序的速度慢上若干量级。因此，这些测试用例本身并不起到关键的作用；相反，它们的作用是提供了启动代码走查和质疑程序员逻辑思路及其设想的手段。在大多数的代码走查中，很多问题是在向程序员提问的过程中发现的，而不是由测试用例本身直接发现的。

与代码检查相同，代码走查参与者所持的态度非常关键。提出的建议应针对程序本身，而不应针对程序员。换句话说，软件中存在的错误不应被视为编写程序的人员自身的弱点。相反，这些错误应被看做是伴随着软件开发的艰难性所固有的。

与代码检查过程中描述的相似，代码走查应该有一个后续过程。同样，代码检查所带来的附带作用（如可以发现易出错的程序区域，通过接触软件错误、编程风格和方法来获得教育等）同样也会发生在代码走查过程中。

3.5 桌面检查

人工查找错误的第三种过程是古老的桌面检查方法。桌面检查可视为由单人进行的代码检查或代码走查：由一个人阅读程序，对照错误列表检查程序，对程序推演测试数据。

对于大多数人而言，桌面检查的效率是相当低的。其中的一个原因是，它是一个完全没有约束的过程。另一个重要的原因是它违反了本书第2章提出的测试原则，即人们一般不能有效地测试自己编写的程序。因此桌面检查最好由其他人而非该程序的编写人员来完成（例如，两个程序员可以相互交换各自的程序，而不是检查自己的程序）。但是即使这样，其效果仍然逊色于代码走查或代码检查。原因在于代码检查和代码走查小组中存在着互相促进的效应。小组会议培养了良性竞争的气氛，人们喜欢通过发现问题来展示自己的能力。而在桌面检查中，由于没有向其他人展示的机会，也就缺乏这个显而易见的良好效应。简而言之，桌面检查胜过没有检查，但其效果远远逊色于代码检查和代码走查。

3.6 同行评审

最后一种人工评审方法与程序测试并无关系（其目标不是为了发现错误），却仍在这里谈到，这是因为它与代码阅读的思想有关。

同行评审是一种依据程序整体质量、可维护性、可扩展性、易用性和清晰性对匿名程序进行评价的技术。该项技术的目的是为程序员提供自我评价的手段。

选出一位程序员来担任这个评审过程的管理员，管理员又会挑选出6～20名参与者（为保持匿名性，6人是最少数量）。这些参与者都应具备相似的背景（例如，不能把Java应用程序员与汇编语言系统程序员编为一组）。要求每名参与者都挑选出两个由自己编写的程序以供评审。其中的一个程序应是参与者自认为能代表其自身能力的最好作品，而另一个则是参与者自认为质量较差的作品。

当所有的程序都收集完毕后，就将这些程序随机分发给参与者。每名参与者拿到4个程序进行评审，其中的两个是“最好”的程序，另外两个则是相对“较差”的程序，但评审人自己并不知道。每名参与者每评审一个程序得花费30分钟，评审完后填写一张评价表。所有4个程序都评审完后，参与者对4个程序的相对质量进行分级。评价表要求评审人用1～10的分值（1代表明确的“是”，10代表明确的“否”），对诸如下面的问题进行回答：

- 程序是否易于理解？
- 高层次的设计是否可见且合理？
- 低层次的设计是否可见且合理？
- 修改此程序对评审者而言是否容易？
- 评审者是否会以编写出该程序而骄傲？

评审人还应给出总的评价和建议的改进意见。

评审结束之后，参与者会收到自己的那两个程序的匿名评价表，此外还会收到一个带统计的总结，说明在所有的程序中其程序的整体和具体得分情况，以及他对其他程序的评价与其他评审人对同一程序打分的比较分析情况。同行评审的目的是让程序员对自身的编程技术进行自我评价。同样，该过程也适用于企业开发和课堂教学环境。

3.7 小结

本章讨论了软件开发人员通常不会考虑到的一种测试形式——人工测试。大多数人认为，因为程序是为了供机器执行而编写的，那么也应由机器来对程序进行测试。这种想法是有问题的。人工测试方法在暴露错误方面是很有成效的。实际上，大多数的软件项目都应使用到以下的人工测试方法：

- 利用错误列表进行代码检查。
- 小组代码走查。
- 桌面检查。
- 同行评审。

另一种人工测试（基于人的测试）就是本章开头提到的可用性测试，这是一种黑盒测试技术，需要测试人员站在最终用户实用的角度来评估软件的可用性程度。这一部分将在本书第7章介绍。

第4章
Chapter 4

测试用例的设计

除了第2章探讨的软件测试的心理学问题以外，软件测试中最重要的因素是设计和生成有效的测试用例。

然而，无论软件测试进行得如何具有创造性、如何完全，也不能保证软件中不存在任何错误。测试用例的设计如此重要，原因在于完全的测试是不可能的，对任何程序的测试必定是不完全的。那么，最显然的测试策略就是努力使测试尽可能完全。

由于时间和成本的约束，软件测试的最关键问题是：

在所有可能的测试用例中，哪个子集最有可能发现最多的错误？

对软件测试用例设计方法的研究为这个问题提供了答案。

一般而言，在所有的方法中效率最低的是随机输入测试，即在所有可能的输入值中随机选取某个子集来对程序进行测试的过程。就发现最多错误的可能性而言，随机选取而产生的测试用例集很少有可能是理想的或接近理想的子集。在本章中，我们将提出一套思考过程，该过程有助于更加睿智地选取测试数据。

本书第2章已经证明穷举的黑盒和白盒测试通常都是不可能的，但同时也建议：将这两种测试的要素组合起来得到一种合理的测试策略。本章将对这种策略进行研究。我们可以通过使用特定的面向黑盒测试的测试用例设计方法，而后使用白盒测试方法对程序的逻辑结构进行检查以补充这些测试用例，借此来设计出一个相当严格的测试。

本章将要讨论的测试方法如下：