第 3 章

软件测试过程与策略

本章概要

软件产品种类繁多，测试过程千变万化，为了能够找到系统中绝大部分的软件缺陷，必须构建各种行之有效的测试方法与策略。

- 软件测试的复杂性和经济性；
- 通过讲述软件测试的整个流程，从而了解单元测试、集成测试、确认测试、系统测试和验收测试等基本测试方法。

3.1 软件测试的复杂性与经济性分析

人们在对软件工程开发的常规认识中，认为开发程序是一个复杂而困难的过程，需要花费大量的人力、物力和时间，而测试一个程序则比较容易，不需要花费太多的精力。这其实是人们对软件工程开发过程理解上的一个误区。在实际的软件开发过程中，作为现代软件开发工业一个非常重要的组成部分，软件测试正扮演着越来越重要的角色。随着软件规模的不断扩大，如何在有限的条件下对被开发软件进行有效的测试正成为软件工程中一个非常关键的课题。

3.1.1 软件测试的复杂性

设计测试用例是一项细致并且需要具备高度技巧的工作，稍有不慎就会顾此失彼，发生不应有的疏漏。下面分析了容易出现问题的根源。

(1) 完全测试是不现实的

在实际的软件测试工作中，由于软件测试情况数量极其巨大，不论采用什么方法，都不可能进行完全彻底的测试。所谓彻底测试，就是让被测程序在一切可能的输入情况下全部执行一遍，通常也称这种测试为“穷举测试”。

彻底测试会引起以下几种问题：①输入量太大；②输出结果太多；③软件执行路径太多；④说明书存在主观性。

由于以上问题的存在，使得在大多数的软件测试过程中，彻底测试几乎是不可能的。

在软件的使用过程中,人们不仅要进行合法的输入,若出现某些意外情况,可能还要发生种种不合法的输入。这样的测试情况可能出现无穷多个,所以测试人员既要测试所有合法的输入,也要对那些不合法但是可能的输入进行测试。

例如,对于常用的画图板程序,如果测试人员受命于使用彻底测试来进行,那么首先要对直线的画图进行测试,把直线中最小的两个相邻点,一个一个地延长直至最大的点,然后是考虑反方向的画图,再是斜线的画图。将所有可能的直线画图全部测试完成之后,还要考虑其他图像的各种画法,一个一个地在理论上将所有可能发生的情况全部测试完毕后,再将可能出现的不同图形的叠加全部实现,当然这里还不包括色彩的运用。按照上述思路一个一个地测试起来,单是合法输入就接近无穷多个,使得在理论上根本无法进行彻底测试。在实际的使用过程中,测试人员还要考虑到随机出现的各种突发情况,例如用户不小心碰到键盘引起某个误操作。Myers 在 1979 年描述了一个只包含 loop 循环和 if 语句的简单程序,可以使用不同的语言将其写成 20 行左右的代码,但是这样简短的语句却有着十万亿条路径。面对这样一个庞大的数字,即便是一个有经验的优秀的软件测试员也需要 10 亿年才能完成全部测试,而且在实际应用中,此类程序是非常有可能出现的。

E. W. Dijkstra 的一句名言对测试的不彻底性作了很好的注解:“程序测试只能证明错误的存在,但不能证明不存在错误”。由于穷举测试工作量太大,实践上行不通,这就注定了一切实际测试都是不彻底的,也就不能够保证被测试程序在理论上不存在遗留的错误。

(2) 软件测试是有风险的

彻底测试的不可行性使得大多数软件在进行测试的时候只能采取非彻底测试,这又意味着存在一种冒险。例如,在使用 Microsoft Office 2003 工具中的 Word 时,可以做这样的一个测试:①新建一个 Word 文档;②在文档中输入汉字“胡”;③设置其字体属性为“隶书”,字号为“初号”,效果为“空心”;④将页面的显示比例设为“500%”。这时在“胡”字的内部会出现“胡万进印”4 个字。类似问题在实际测试中如果不使用彻底测试是很难发现的,而如果在软件投入市场时才发现则修复代价就会非常高。这就会产生一个矛盾:软件测试员不能做到完全的测试,不完全测试又不能证明软件的百分之百的可靠。那么如何在这两者的矛盾中找到一个相对的平衡点呢?

从图 3-1 所示的最优测试量示意图可以观察到,当软件缺陷降低到某一数值后,随着测试量的不断上升软件缺陷并没有明显地下降。这是软件测试工作中需要注意的重要问题。如何把巨大的测试数据量减少到可以控制的范围,如何针对风险做出最明智的选择是软件测试人员必须能够把握的关键问题。

图 3-1 所示的最优测试量示意图说明了发现软件缺陷数量和测试量之间的关系,随着测试量的增加,测试成本将呈几何数级上升,而软件缺陷降低到某一数值之后将没有明显的变化,最优测量值就是这两条曲线的交点。

针对问题没有十全十美的解决办法,采取最优测试量只是在两者中的一种妥协。然而糟糕的是矛盾还不止于此,测试量会随着时间的推移而发生改变。在当今竞争激烈的市场里,争取时间可能是制胜的关键,这本身就使软件的开发与测试出现矛盾。使情况更加复杂的是,当一种新的技术或者新的标准出现时,可能人们会对软件是否十全十美并不

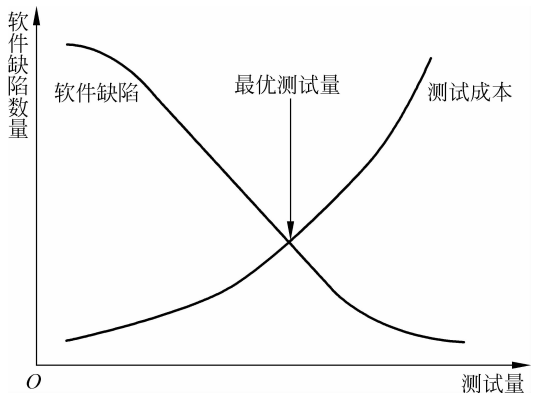

图 3-1　最优测试量示意图

在意了。在这种情况下，要进行多长时间的测试就更是一个值得商榷的问题。

微软公司在研制 Windows 操作系统的第一个版本时已经落后于对手了，但是为了能抢得第一套应用于 PC 的图形界面操作系统这一市场先机，微软公司一面大打广告宣传，一面在公司里加班加点。为了能尽快进行产品的发布，在没有进行可靠的测试验证的情况下，微软公司就公布了自己的 Windows 操作系统。虽然这时的 Windows 操作系统漏洞百出，但是仍然赢得了绝大部分的市场份额。反观微软公司的竞争对手，虽然对产品进行了完善的测试与验证，但这时候已经没有人来关注他们的产品了，大家都在兴致勃勃地讨论着 Windows 操作系统，竞争公司最终退出了这一市场。

当然，上面的例子只是在市场初期的特殊现象。如果市场分配格局已经建立起来，那么就应该针对合适的目标加大测试量，提高产品质量。但是在不同的市场时期，如何决定测试量的多少对于一个软件开发公司仍然是一个十分重要的课题，这不仅涉及软件技术知识，还要考虑潜在用户的心理分析和商品运营规律等因素。

(3) 杀虫剂现象

1990 年，Boris Beizer 在其编著的《Software Testing Techniques》(第 2 版)中提到了“杀虫剂怪事”一词。同一种测试工具或方法用于测试同一类软件越多，则被测试软件对测试的免疫力就越强。这与农药杀虫是一样的，老用一种农药，则害虫就有了免疫力，农药就失去了作用。

由于软件开发人员在开发过程中可能碰见各种各样的主客观因素，再加上不可预见的突发性事件，所以再优秀的软件测试员采用一种测试方法或者工具也不可能检测出所有的缺陷。为了克服被测试软件的免疫力，软件测试员必须不断编写新的测试程序，对程序的各个部分不断地进行测试，以避免被测试软件对单一的测试程序具有免疫力而使软件缺陷不被发现。这就对软件测试人员的素质提出了很高的要求。

(4) 缺陷的不确定性

在软件测试中还有一个让人不容易判断的现象是缺陷的不确定性，即并不是所有的软件缺陷都需要被修复。对于究竟什么才算是软件缺陷是一个很难把握的标准，在任何一本软件测试的书中都只能给出一个笼统的定义。实际测试中需要把这一定义根据具体

的被测对象明确化。即使这样,具体的测试人员对软件系统的理解不同,还是会出现不同的标准。

当确定是软件缺陷时,若出现以下情况,软件缺陷也不能被修复。

① 修复的风险太大。软件在编译期间,本身是一个很脆弱的系统。由于在整个软件系统中各个模块之间有着千丝万缕的联系,使得单一修复某一段代码可能引起大量的未知的缺陷。所以在某些非常时期不修复反而是最保险的做法。

② 时间不够。在商业社会中,当部分软件缺陷没有足够的时间修复,就只能在说明书中列出可能出现的缺陷。

③ 不会引起大的问题。为了防止整个系统由于局部修复而出现某些问题,在特殊情况下,不常出现的小问题可以暂时忽略。

④ 可以理解成新的功能。某些特殊的缺陷有时从另一个方面看可以理解成一种新的功能。这是大多数商务软件在处理一些特殊缺陷时采取的做法。

3.1.2 软件测试的经济性

软件测试的经济性有两方面体现:一是体现在测试工作在整个项目开发过程中的重要地位;二是体现在应该按照什么样的原则进行测试,以实现测试成本与测试效果的统一。软件工程的总目标是充分利用有限的人力和物力资源,高效率、高质量地完成测试。结合上一节关于完全测试具有不可行性的介绍,就可以理解为什么要在测试量与测试成本的曲线中选取最优测试点。为了降低测试成本,在选择测试用例时要遵守以下原则。

(1) 被测对象的测试等级应该根据被测对象在整个软件开发项目中的重要地位和一旦发生故障造成的损失情况来综合分析。

(2) 要制订科学有效的测试策略。在保证能够尽可能多地发现软件缺陷的前提下,尽量少地使用测试用例。如何找到最优测试点,掌握好测试用量是至关重要的。一位有经验的软件管理人员在谈到软件测试时曾这样说过:“不充分的测试是愚蠢的,而过度的测试是一种罪孽”。测试不足意味着让用户承担隐藏错误带来的危险,过度测试则会浪费许多宝贵的资源。

测试是软件生存期中费用消耗最多的环节。测试费用除了测试的直接消耗外,还包括其他的相关费用。影响测试费用的主要因素有以下几点。

(1) 软件面向的目标用户

软件产品需要达到的标准决定了测试的数量。对于那些至关重要的系统必须要进行更多的测试。一个在 Boeing 757 上运行的系统应该比一个用于公共图书馆中检索资料的系统需要更多的测试。一个用来控制银行证券实时交易的系统应该比一个简单的网上实时交流系统具有更大的可靠性与可信度。一个用于国防的大型安全关键软件的开发组比一个网络游戏软件开发组要有苛刻得多的查找错误方面的要求。

(2) 可能出现的用户数量

一个系统的目标用户数量的多少也在很大程度上影响了测试必要性的程度,这主要是由于用户团体在经济方面的影响。一个在全世界范围内有几千个用户的系统肯定比一个只在办公室中运行的有两三个用户的系统需要更多的测试。如果出现问题的话,前一个系统的经济影响肯定比后一个系统大。另外,在错误处理的分配上,所需花费代价的差

别也很大。如果在内部系统中发现了一个严重的错误，处理错误的费用就会相对少一些。如果要处理一个遍布全世界的错误则要花费相当大的财力和精力，而且还会给开发公司造成严重的信誉危机和潜在用户的流失。

(3) 潜在缺陷造成的影响

在考虑测试的必要性时，还需要将系统中所包含的信息价值考虑在内。例如，一个支持许多家大银行或众多证券交易所的客户机/服务器系统中一定含有经济价值非常高的内容。由于银行证券系统的特殊性，一旦出现问题，影响的将不仅是银行或证券公司，将波及所有与银行或证券公司有业务往来的公司或个人，后果将非常恶劣。很显然，这样的大型系统和其他单一的小型应用系统相比，需要进行更多的测试。这两种系统的用户都希望得到高质量、无错误的系统，但是前一种系统的影响比后一种要大得多。因此应该从经济方面考虑，根据投入与经济价值相对应的时间和金钱去进行测试。

(4) 开发机构的业务能力

一个没有标准和缺少经验的开发机构很可能开发出充满错误的系统。在一个建立了标准和有很多经验的开发机构中开发出来的系统中的错误将会少很多。然而，那些需要进行大幅度改善的机构反而不大可能认识到自身的弱点。那些需要更加严格的测试过程的机构往往是最不可能进行这一活动的。在许多情况下，机构的管理部门并不能真正地理解开发一个高质量的系统的好处，反而是那些拥有很多经验和建立了严格标准的开发机构更加重视软件测试的重要性。

3.1.3　软件测试的充分性准则

软件测试的充分性准则有以下几点。

(1) 对任何软件都存在有限的充分测试集合。

(2) 当一个测试的数据集对于一个被测的软件系统的测试是充分的，那么再多增加一些测试数据仍然是充分的。这一特性称为软件测试的单调性。

(3) 即使对软件所有成分都进行了充分的测试，也并不意味着整个软件的测试已经充分了。这一特性称为软件测试的非复合性。

(4) 即使对一个软件系统整体的测试是充分的，也并不意味着软件系统中各个成分都已经充分地得到了测试。这个特性称为软件测试的非分解性。

(5) 软件测试的充分性与软件的需求、软件的实现都相关。

(6) 软件测试的数据量正比于软件的复杂度。这一特性称为软件测试的复杂性。

(7) 随着测试次数的增加，可以检查出软件缺陷的几率随之不断减少，软件测试具有回报递减率。

3.1.4　软件测试的误区

随着软件产业工业化、模块化地发展，一个软件开发组中软件测试人员的重要性不断突出。在国外，很多著名企业早已对软件测试工作十分重视。例如著名的微软公司，其软件测试人员与开发人员的比例已经达到 2∶1。可见软件测试对于一个软件开发项目的成功与否具有十分重要的意义。但是在实际的项目开发与管理中仍然存在很多管理上或者技术上的误区。

(1) 期望用测试自动化代替大部分人工劳动

通过应用自动化测试工具能够帮助完成部分重复枯燥的手工作业,但自动化测试工具不能完全代替人工测试。一般来讲,产品化的软件更适于功能测试的自动化。由标准模块组装的系统更好,因为其功能稳定,界面变化不大。

对于测试自动化的使用可以按以下规则来进行:自动化20%的测试用例,用于覆盖80%的用户操作密集的功能和核心商业逻辑(例如工资计算准确度要求高,虽然每月才执行一次)。实现功能测试自动化来完成重复枯燥的回归测试任务,引入性能测试自动化工具来改善测试的广度和深度。测试自动化会带来一点好处,毕竟机器和脚本是客观的,它总是会完成测试员分配的所有任务,而没有半点遗漏,从而有助于测试员真正掌握和控制回归测试的覆盖率。

(2) 忽视需求阶段的参与

在某些公司,产品原始需求文档本来就不是很完善,从市场调研人员到项目经理、开发经理、Leader,再到具体Coding的程序员,每一层之间的传递都有可能存在需求理解上的偏差。让测试人员参与需求阶段的工作,可以在一定程度上起到双保险,更好地杜绝需求和实现之间差异的发生。软件测试工作同时兼顾了"证明软件的实现和需求是一致的"和"验证软件在某些情况下可能会产生问题"两个方面。因此,测试人员对需求的理解就从另一个角度影响了整个测试工作的可靠性和效率。测试人员和开发人员同时、同等地从上游获得需求,并持有自己的理解,可以排除部分功能实现和需求错位的问题。

(3) 软件测试是技术要求不高的岗位

单从目前用人最多的黑盒功能测试岗位来说,测试人员对计算机技术的要求或许不是很高。实际上,测试人员除了逻辑思维、沟通能力等自身素质外,技能暂且可以分为两种:①行业知识,如丰富的财务或ERP实施经验;②计算机技术,如计算机语言设计和软件项目开发经验。

好的测试人员,不但要不懈地执行常规的测试任务,更要有严谨的态度和缜密的思维,去覆盖更多的"可能",发现别人很难找到的软件缺陷。要利用自己丰富的行业经验,判断需求到系统功能的实现是否合理。要站在一定高度对软件框架、设计方法、项目管理等提出合理的建议。

所有这些,加上软件测试管理相关的其他技术(如配置管理等),对于一名合格的软件测试人员的素质要求是很高的。

3.2 软件测试流程

3.2.1 软件开发的V模型

1. V模型

软件测试是有阶段性的,而软件测试的流程与软件设计周期究竟是什么样的关系呢?软件开发流程的V模型是一个广为人知的模型,如图3-2所示。在V模型中,从左到右描述了基本的开发过程和测试行为,为软件的开发人员和测试管理者提供了一个极为简单的框架。V模型的价值在于它非常明确地标明了测试过程中存在的不同级别,并且清

楚地描述了这些测试阶段和开发过程期间各阶段的对应关系。

在 V 模型中各个测试阶段的执行流程如下：单元测试是基于代码的测试，最初由开发人员执行，以验证其可执行程序代码的各个部分是否已达到了预期的功能要求；集成测试验证了两个或多个单元之间的集成是否正确，并且有针对性地对详细设计中所定义的各单元之间的接口进行检查；在单元测试和集成测试完成之后，系统测试开始用客户环境模拟系统的运行，以验证系统是否达到了在概要设计中所定义的功能和性能；最后，在技术部门完成了所有测试工作之后，由业务专家或用户进行验收测试，以确保产品能真正符合用户业务上的需要。

图 3-2 所示描绘出了各个测试环节在整个软件测试工作中的相互联系与制约关系。

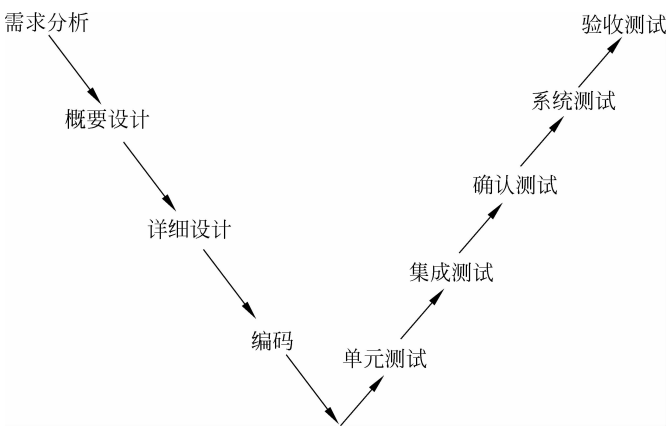

图 3-2　V 模型示意图

2. 软件测试过程

软件测试过程按各测试阶段的先后顺序可分为单元测试、集成测试、确认(有效性)测试、系统测试和验收(用户)测试 5 个阶段，如图 3-3 所示。

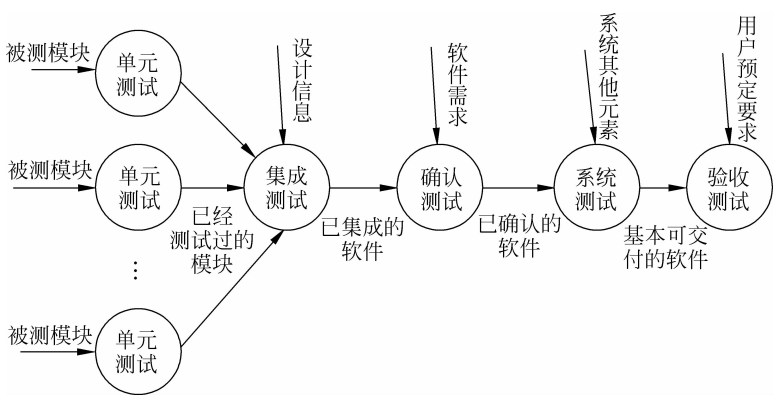

图 3-3　测试各阶段示意图

(1) 单元测试：测试执行的开始阶段。测试对象是每个单元。测试目的是保证每个模块或组件能正常工作。单元测试主要采用白盒测试方法，检测程序的内部结构。

(2) 集成测试：也称组装测试。在单元测试基础上，对已测试过的模块进行组装和集成测试。集成测试的目的是检验与接口有关的模块之间的问题。集成测试主要采用黑盒测试方法。

(3) 确认测试：也称有效性测试。在完成集成测试后，验证软件的功能和性能及其他特性是否符合用户要求。测试目的是保证系统能够按照用户预定的要求工作。确认测试通常采用黑盒测试方法。

(4) 系统测试：在完成确认测试后，为了检验它能否与实际环境(如软硬件平台、数据和人员等)协调工作，还需要进行系统测试。可以说，系统测试之后，软件产品基本满足开发要求。

(5) 验收测试：测试过程的最后一个阶段。验收测试主要突出用户的作用，同时软件开发人员也应该参与进去。

软件测试阶段的输入信息包括两类：①软件配置：指测试对象，通常包括需求说明书、设计说明书和被测试的源程序等；②测试配置：通常包括测试计划、测试步骤、测试用例以及具体实施测试的测试程序、测试工具等。

对测试结果与预期的结果进行比较之后，即可判断是否存在错误，决定是否进入排错阶段，进行调试任务。对修改以后的程序要进行重新测试，因为修改可能会带来新的问题。

通常根据出错的情况得到出错率来预估被测软件的可靠性，这将对软件运行后的维护工作有重要价值。

3.2.2 单元测试

1. 单元测试的定义

单元测试(Unit Testing)是对软件基本组成单元进行的测试。单元测试的对象是软件设计的最小单位——模块。很多人将单元的概念误解为一个具体函数或一个类的方法，这种理解并不准确。作为一个最小的单元应该有明确的功能定义、性能定义和接口定义，而且可以清晰地与其他单元区分开来。一个菜单、一个显示界面或者能够独立完成的具体功能都可以是一个单元。某种意义上单元的概念已经扩展为组件(Component)。

单元测试通常是开发者编写的一小段代码，用于检验被测代码的一个很小的、很明确的功能是否正确。通常而言，一个单元测试是用于判断某个特定条件(或者场景)下某个特定函数的行为。例如，可以把一个很大的值放入一个有序表中去，然后确认该值出现在有序表的尾部，或者可以从字符串中删除匹配某种模式的字符，然后确认字符串确实不再包含这些字符了。单元测试是由程序员自己来完成，最终受益的也是程序员自己。可以这么说，程序员有责任编写功能代码，同时也就有责任为自己的代码编写单元测试。执行单元测试，就是为了证明这段代码的行为和期望的一致。打一个比喻，单元测试就像工厂在组装一台电视机之前，会对每个元件都进行测试。其实程序员每天都在做单元测试，你写了一个函数，除了极简单的外，总是要执行一下，看看软件的功能是否正常，有时还要想办法输出些数据，比如弹出信息窗口什么的，这也是单元测试，一般把这种单元测试称为临时单元测试。对于程序员来说，如果养成了对自己写的代码进行单元测试的习惯，不但可以写出高质量的代码，而且还能提高编程水平。

2. 单元测试的目标

单元测试的主要目标是确保各单元模块被正确地编码。单元测试除了保证测试代码的功能性，还需要保证代码在结构上具有可靠性和健全性，并且能够在所有条件下正确响应。进行全面的单元测试，可以减少应用级别所需的工作量，并且彻底减少系统产生错误的可能性。如果手动执行，单元测试可能需要大量的工作，而自动化测试会提高测试效率。

3. 单元测试的内容

单元测试的主要内容有：①模块接口测试；②局部数据结构测试；③独立路径测试；④错误处理测试；⑤边界条件测试。

如图 3-4 所示，这些测试都作用于模块，共同完成单元测试任务。

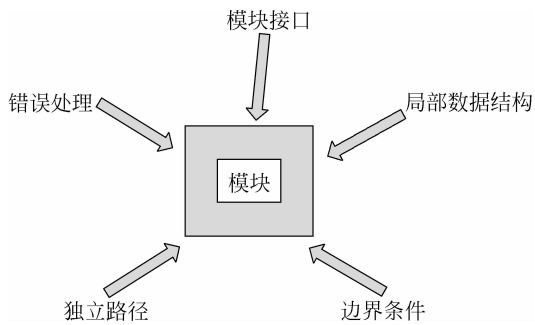

图 3-4　单元测试内容

模块接口测试：对通过被测模块的数据流进行测试。为此，对模块接口（包括参数表、调用子模块的参数、全程数据、文件输入/输出操作）都必须进行检查。

局部数据结构测试：设计测试用例检查数据类型说明、初始化、默认值等方面的问题，还要查清全程数据对模块的影响。

独立路径测试：选择适当的测试用例，对模块中重要的执行路径进行测试。对基本执行路径和循环进行测试可以发现大量的路径错误。

错误处理测试：检查模块的错误处理功能是否包含错误或缺陷。例如，是否拒绝不合理的输入；出错的描述是否难以理解、是否对错误定位有误、是否出错原因报告有误、是否对错误条件的处理不正确；在对错误处理之前错误条件是否已经引起系统的干预等。

边界条件测试：要特别注意数据流、控制流中刚好等于、大于或小于确定的比较值时出错的可能性。对这些地方要仔细地选择测试用例，认真加以测试。此外，如果对模块运行时间有要求的话，还要专门进行关键路径测试，以确定最坏情况下和平均意义下影响模块运行时间的因素。这类信息对进行性能评价是十分有用的。

4. 单元测试的步骤

通常单元测试在编码阶段进行。当源程序代码编制完成，经过评审和验证，确认没有语法错误之后，就开始进行单元测试的测试用例设计。利用设计文档，设计可以验证程序功能、找出程序错误的多个测试用例。对于每一组输入，应有预期的正确结果。

模块并不是一个独立的程序,在考虑测试模块时,同时要考虑它和外界的联系,用一些辅助模块去模拟与被测模块相关联的其他模块。这些辅助模块可分为两种。

(1) 驱动模块(Driver):相当于被测模块的主程序。它接收测试数据,把这些数据传送给被测模块,最后输出实测结果。

(2) 桩模块(Stub):用以代替被测模块调用的子模块。桩模块可以做少量的数据操作,不需要把子模块所有功能都带进来,但也不允许什么事情都不做。

被测模块、与它相关的驱动模块以及桩模块共同构成了一个"测试环境",如图3-5所示。

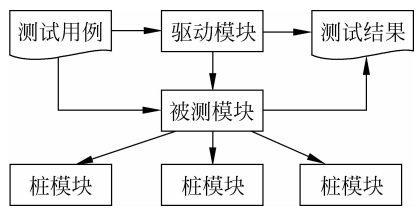

图3-5 单元测试环境

如果一个模块要完成多种功能,并且以程序包或对象类的形式出现,例如Ada中的包,Modula中的模块,C++中的类,这时可以将模块看成由几个小程序组成。对其中的每个小程序先进行单元测试要做的工作,再对关键模块做性能测试。对支持某些标准规程的程序,更要着手进行互联测试。有人把这种情况特别称为模块测试,以区别单元测试。

5. 采用单元测试的原因

程序员编写代码时,一定会反复调试保证其能够编译通过。如果是编译没有通过的代码,没有任何人会愿意将其交付给自己的老板。但代码通过编译,只是说明了它的语法正确,程序员却无法保证它的语义也一定正确。没有任何人可以轻易承诺这段代码的行为一定是正确的。单元测试这时会为此做出保证。

编写单元测试就是用来验证这段代码的行为是否与软件开发人员期望的一致。有了单元测试,程序员可以自信地交付自己的代码,而没有任何的后顾之忧。

什么时候进行单元测试呢?单元测试进行得越早越好。早到什么程度呢?开发理论讲究TDD,即测试驱动开发,先编写测试代码,再进行开发。在实际的工作中,可以不必过分强调先干什么后干什么,重要的是高效且感觉舒适。从实际开发经验来看,先编写产品函数的框架,然后编写测试函数,针对产品函数的功能编写测试用例,然后编写产品函数的代码,每写一个功能点都运行并进行测试,随时补充测试用例。所谓先编写产品函数的框架,是指先编写函数框架的实现,有返回值的随便返回一个值,编译通过后再编写测试代码。这时,函数名、参数表、返回类型都应该确定下来了,所编写的测试代码以后需修改的可能性比较小。

由谁来完成单元测试呢?单元测试与其他测试不同,单元测试可看作是编码工作的一部分,应该由程序员完成。也就是说,经过了单元测试的代码才是已完成的代码,提交产品代码时也要同时提交测试代码。测试部门可以进行一定程度的审核。在一种传统的结构化编程语言中,如C语言中,要进行测试的单元一般是函数或子过程。在像C++这样的面向对象的语言中,要进行测试的基本单元是类。对Ada语言来说,开发人员可以选择是在独立的过程和函数的级别上,还是在Ada包的级别上进行单元测试。单元测试的原则同样被扩展到第四代语言(4GL)的开发中,在这里基本单元被典型地划分为一个菜单或显示界面。单元测试是作为无错编码的一种辅助手段,在一次性的开发过程中使