第 3 章　软件测试过程与方法

本章目标

- 掌握软件测试的过程
- 掌握软件测试与开发的关系
- 熟悉单元测试
- 熟悉集成测试
- 熟悉确认测试
- 熟悉系统测试
- 熟悉验收测试

本章单词

unit test：_____　　integration test：_____

system test：_____　　acceptance test：_____

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。

(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，进行集成测试。测试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。

图 3-1 展示了在不同的测试阶段，测试的方法及内容都不同。

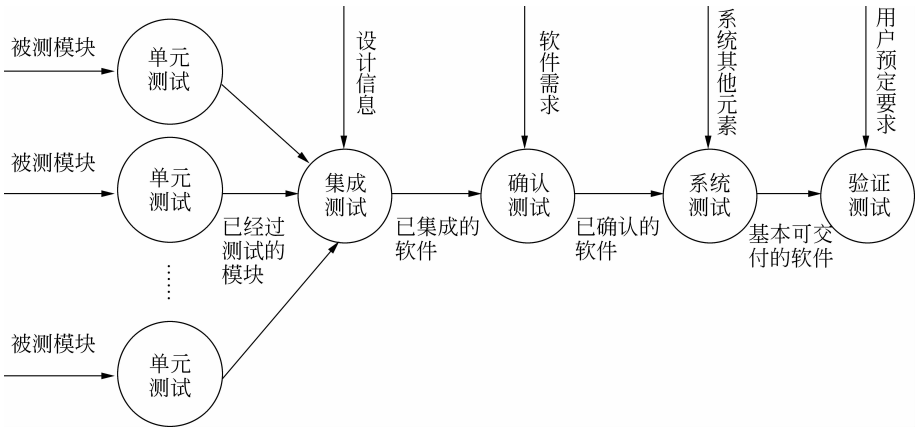

图 3-1　软件测试过程

3.1　单元测试

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。

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。从某种意义上单元的概念已经扩展为组件(component)。

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,自动化测试会提高测试效率。

3. 单元测试的内容

如图 3-2 所示,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。

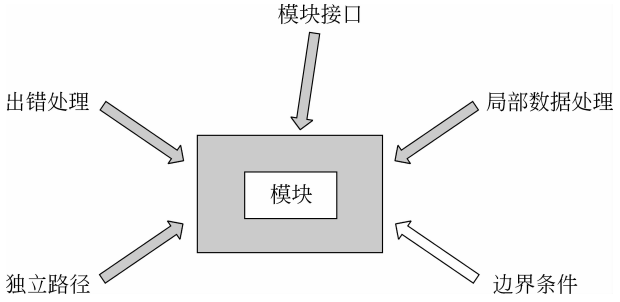

图 3-2　单元测试任务

这些测试都作用于模块,共同完成单元测试任务。

(1) 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/输出操作都必须检查。

(2) 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。

(3) 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。基本路径测试和循环测试可以发现大量的路径错误,是最常用且最有效的测试技术。

(4) 错误处理测试:检查模块的错误处理功能是否包含有错误或缺陷。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。

(5) 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，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。

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。

模块接口测试中的被测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模块。这些辅助模块可分为以下两种。

(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，但不允许什么事情也不做。

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 3-3 所示。

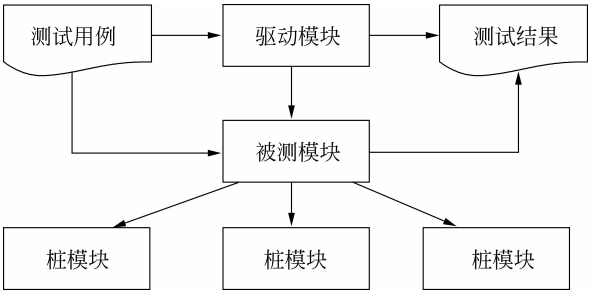

图 3-3　单元测试环境

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 语言中的包，MODULA 语言中的模块，C++ 语言中的类，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，对关键模块还要做性能测试。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。

3.2　集成测试

所有的软件项目都不能摆脱系统集成这个阶段。不管采用什么开发模式，具体的开发工作总得从一个一个的软件单元做起，软件单元只有经过集成才能形成一个有机的整体。

1. 集成测试的定义

在完成单元测试的基础上，需要将所有模块按照设计要求组装成为系统。这时需要考虑以下问题：

(1) 在把各个模块连接起来的时候，穿越模块接口的数据是否会丢失；

(2) 一个模块的功能是否会对另一个模块的功能产生不利的影响；

(3) 各个子功能组合起来,能否达到预期要求的父功能;

(4) 全局数据结构是否有问题;

(5) 单个模块的误差累积起来,是否会放大,从而达到不能接受的程度;

(6) 单个模块的错误是否会导致数据库错误。

集成测试(integration testing)是介于单元测试和系统测试之间的过渡阶段,与软件开发计划中的软件概要设计阶段相对应,是单元测试的扩展和延伸。

集成测试的定义是根据实际情况对程序模块采用适当的集成测试策略组装起来,对系统的接口以及集成后的功能进行正确校验的测试工作。

2. 集成测试的层次

软件的开发过程是一个从需求分析到概要设计、详细设计以及编码实现的逐步细化的过程,那么单元测试到集成测试再到系统测试就是一个逆向求证的过程。集成测试内部对于传统软件和面向对象的应用系统有两种层次的划分。

对于传统软件来讲,可以把集成测试划分为三个层次:

(1) 模块内集成测试;

(2) 子系统内集成测试;

(3) 子系统间集成测试。

对于面向对象的应用系统来说,可以把集成测试分为两个阶段:

(1) 类内集成测试;

(2) 类间集成测试。

3. 集成测试的模式

选择什么方式把模块组装起来形成一个可运行的系统,直接影响到模块测试用例的形式、所用测试工具的类型、模块编号的次序和测试的次序、生成测试用例的费用和调试的费用。集成测试模式是软件集成测试中的策略体现,其重要性是明显的,直接关系到软件测试的效率、结果等,一般是根据软件的具体情况来决定采用哪种模式。通常,把模块组装成为系统的测试方式有以下两种。

(1) 一次性集成测试方式(no-incremental integration):一次性集成测试方式也称作非增值式集成测试。先分别测试每个模块,再把所有模块按设计要求放在一起结合成所需要实现的程序。

(2) 增值式集成测试方式:把下一个要测试的模块同已经测好的模块结合起来进行测试,测试完毕,再把下一个应该测试的模块结合进来继续进行测试。在组装的过程中边连接边测试,以发现连接过程中产生的问题。通过增值逐步组装成为预先要求的软件系统。增值式集成测试方式有以下三种:

① 自顶向下增值测试方式(top-down integration);

② 自底向上增值测试方式(bottom-up integration);

③ 混合增值测试方式(modified top-down integration)。

(3) 一次性集成测试方式与增值式集成测试方式的比较:增值式集成方式需要编写的软件较多,工作量较大,花费的时间较多。一次性集成方式的工作量较小;增值式集成方式发现问题的时间比一次性集成方式早;增值式集成方式比一次性集成方式更容易判断出问题的所在,因为出现的问题往往和最后加进来的模块有关;增值式集成方式测试更为彻底;