第 3 章

软件测试的执行阶段

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。本章通过讲述软件测试的整个流程，帮助读者了解单元测试、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 3-1 所示。

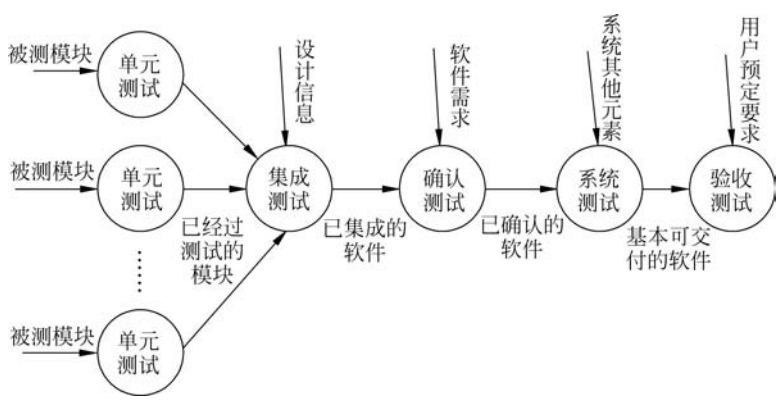

图 3-1 软件测试各阶段示意图

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。

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。

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。

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。经过系统测试之后，软件产品基本能满足开发要求。

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以后，即可判断程序是否存在错误，从而决定是否进入排错阶段，进行调试任务。另外，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。通常根据软件测试的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具有重要价值。

3.2 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。从某种意义上讲，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，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。单元测试由程序员自己完成，最终受益的也是程序员自己。程序员有责任编写功能代码，同时也有责任为自己的代码编写单元测试。执行单元测试是为了证明这段代码的行为和期望的一致。像工厂在组装一台电视机之前，会对每个元件都进行测试一样，需要对软件的基本组成单元进行测试，这就是单元测试。其实，程序员每天都在做单元测试。例如，程序员写完一个函数总是要执行一下，以确认函数的功能是否正常；有时，还要输出一些提示信息，比如弹出信息窗口等。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外，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，可以减少应用级别测试所需的工作量，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

单元测试的主要内容有以下几个方面。

（1）模块接口测试：对通过被测模块的数据流进行测试。为此，对模块接口来说，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。

（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。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。

（4）错误处理测试：检查模块的错误处理功能是否有错误或缺陷。例如，是否拒绝不合理的输入；出错的描述是否难以理解、对错误定位是否有误、出错原因报告是否有误、对错误条件的处理是否有误；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：在测试中，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情况下影响模块运行时间的因素。这类信息对软件进行性能评价十分有用。

上述这些测试都作用于模块，共同完成单元测试任务，如图 3-2 所示。

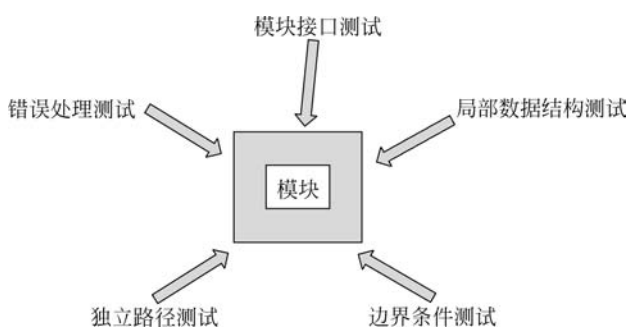

图 3-2　单元测试任务

4. 单元测试的步骤

单元测试通常在编码阶段进行。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以下两种。

(1) 驱动模块(Drive Module):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 Module):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。

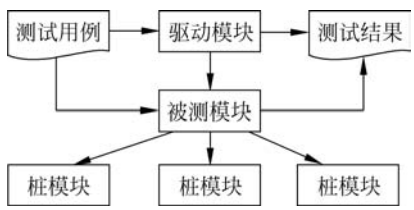

图3-3 单元测试环境

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中的包,Module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,对关键模块还要做性能测试。对支持某些标准规程的程序,还需进行互联测试。有人把这种情况称为模块测试,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通过编译。但代码通过编译,只能说明代码的语法正确,程序员无法保证代码的语义也一定正确。没有任何人可以轻易承诺某段代码的行为一定是正确的。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,而没有任何后顾之忧。

单元测试越早越好。开发理论讲究(Test-Driven Development,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。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,是指先编写函数空的实现,有返回值的任意返回一个值,编译通过后再编写测试代码。这时,函数名、参数表、返回类型都已经确定,所编写的测试代码以后需修改的可能性比较小。

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,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。在传统的结构化编程语言中,比如C语言,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或函数,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(Fourth-Generation Language,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用。另外,单元测试必须是可重复的,无论是在软件修

改，还是移植到新的运行环境的过程中。因此，所有的测试都必须在整个软件系统的生命周期中进行维护。

通过单元测试，测试人员可以验证开发人员所编写的代码是按照先前设想的方式进行的，输出结果符合预期值，这就实现了单元测试的目的。与后面阶段的测试相比，单元测试创建简单，维护容易，并且可以更方便地进行重复。《实用软件度量》(Capers Jones，McGraw-Hill，1991)一书中列出了各测试阶段在准备测试、执行测试和修改缺陷方面所花费的时间(以一个功能点为基准)，单元测试的成本效率大约是集成测试的两倍、系统测试的三倍，如图 3-4 所示。术语域测试是指软件在投入使用后，针对某个领域所做的所有测试活动。

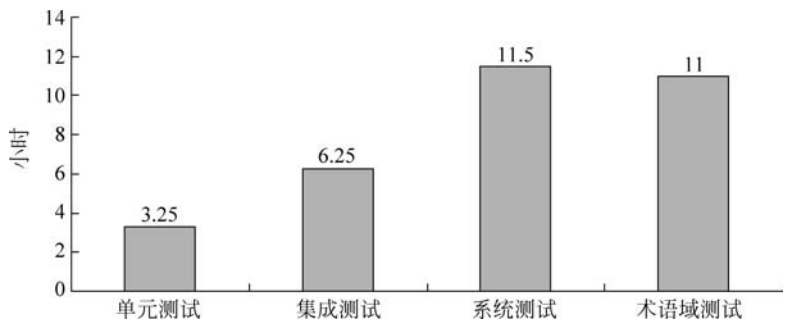

图 3-4　各测试阶段发现缺陷花费的时间

3.3　集成测试

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。集成测试也称为综合测试。实践表明，软件的一些模块能够单独地工作，但并不能保证连接之后也能正常工作。程序在某些局部反映不出来的问题，在全局上有可能暴露出来，影响软件功能的实现。所以，集成测试是针对程序整体结构的测试。

2. 集成测试的层次

软件的开发过程是一个从需求到概要设计、详细设计以及编码的逐步细化的过程,那么单元测试到集成测试再到系统测试则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:模块内集成测试、子系统内集成测试和子系统间集成测试。

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:类内集成测试和类间集成测试。

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,具有非常重要的作用,直接关系到软件测试的效率、结果等,一般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两种。

1) 一次性集成测试方式

一次性集成测试方式(No-incremental Integration)也称非增值式集成测试。测试时,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。

一次性集成测试方式的实例如图3-5所示。整个系统结构如图3-5(a)所示,共包含6个模块,具体测试过程如下。

对模块B进行单元测试。为模块B配备驱动模块D1,用来模拟模块A对B的调用;为模块B配备桩模块S1,用来模拟模块C被B调用,如图3-5(b)所示。

对模块D进行单元测试。为模块D配备驱动模块D3以及桩模块S2,如图3-5(d)所示。

对模块C、E、F分别进行单元测试。为模块C、E、F分别配备驱动模块D2、D4、D5,如图3-5(c)、图3-5(e)、图3-5(f)所示。

对模块A进行单元测试。为主模块A配备三个桩模块S3、S4、S5,如图3-5(g)所示。

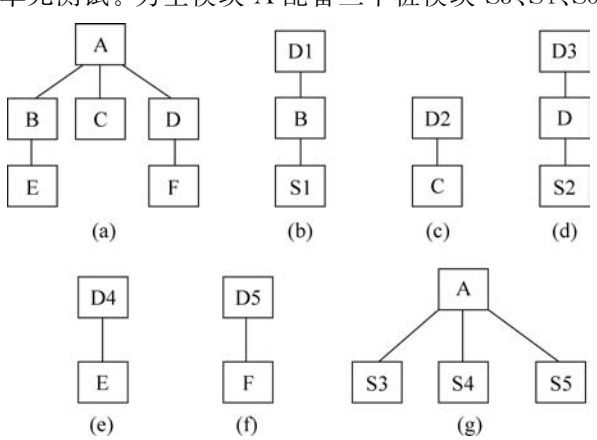

图3-5 一次性集成测试方式实例

在将模块 A、B、C、D、E 分别进行了单元测试之后，再一次性进行集成测试。

测试结束。

2）增值式集成测试方式

把下一个要测试的模块同已经测试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三种。

（1）自顶向下增值测试方式（Top-down Integration）。主控模块作为测试驱动，所有与主控模块直接相连的模块作为桩模块；根据集成的方式（深度或广度），每次用一个模块把从属的桩模块替换成真正的模块；在每个模块被集成时，都必须已经进行了单元测试；进行回归测试以确定集成新模块后没有引入错误。这种组装方式将模块按系统程序结构，沿着控制层次自顶向下进行组装。自顶向下的增值方式在测试过程中较早地验证了主要的控制和判断点。选用按深度方向组装的方式，可以首先实现和验证一个完整的软件功能。

按照深度优先方式遍历的自顶向下增值的集成测试实例如图 3-6 所示。具体测试过程如下。

在树状结构图中，按照先左后右的顺序确定模块集成路线。

先对顶层的主模块 A 进行单元测试。就是对模块 A 配以桩模块 S1、S2 和 S3，用来模拟它所实际调用的模块 B、C、D，然后进行测试，如图 3-6(a)所示。

用实际模块 B 替换掉桩模块 S1，与模块 A 连接，再对模块 B 配以桩模块 S4，用来模拟模块 B 对 E 的调用，然后进行测试，如图 3-6(b)所示。

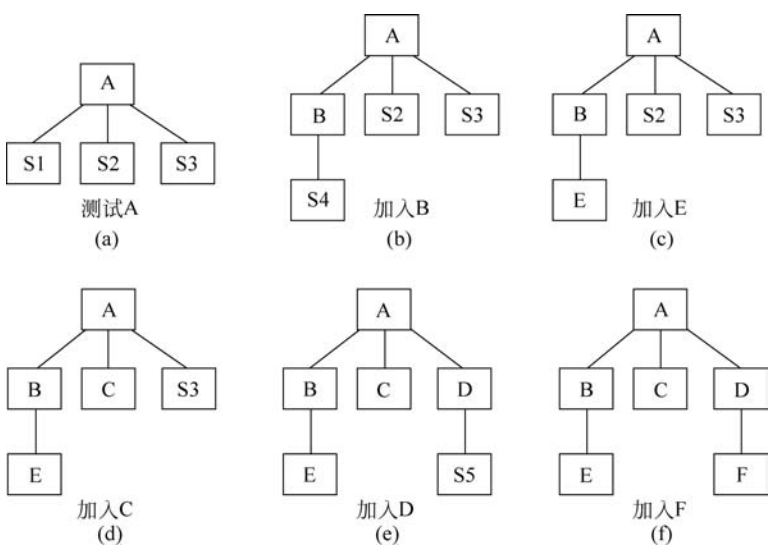

图 3-6　自顶向下增值测试方式实例

将模块 E 替换掉桩模块 S4 并与模块 B 相连，然后进行测试，如图 3-6(c)所示。

判断模块 E 没有叶子节点，即以 A 为根节点的树状结构图中的最左侧分支深度遍历

结束。转向下一个分支。

模块C替换掉桩模块S2,连到模块A上,然后进行测试,如图3-6(d)所示。

判断模块C没有桩模块,转到树状结构图的最后一个分支。

模块D替换掉桩模块S3,连到模块A上,同时给模块D配以桩模块S5,来模拟其对模块F的调用,然后进行测试,如图3-6(e)所示。

去掉桩模块S5,替换成实际模块F连接到模块D上,然后进行测试,如图3-6(f)所示。

通过上述步骤完成对树状结构图的完全测试,测试结束。

(2) 自底向上增值测试方式(Bottom-up Integration)。组装从最底层的模块开始,组合成一个构件,用以完成指定的软件子功能。编制驱动程序,协调测试用例的输入与输出;测试集成后的构件;按程序结构向上组装测试后的构件,同时除掉驱动程序。这种组装的方式是从程序模块结构的最底层的模块开始组装和测试。因为模块是自底向上进行组装,对于一个给定层次的模块,它的子模块(包括子模块的所有下属模块)已经组装并测试完成,所以不再需要桩模块。在模块的测试过程中需要从子模块得到的信息可以通过直接运行子模块获得。

按照自底向上增值的集成测试实例如图3-7所示。首先,对处于树状结构图中叶子节点位置的模块E、C、F进行单元测试,如图3-7(a)~(c)所示,分别配以驱动模块D1、D2和D3,用来模拟模块B、模块A和模块D对它们的调用。然后去掉驱动模块D1和D3,替换成模块B和D分别与模块E和F相连,并且设立驱动模块D4和D5进行局部集成测试,如图3-7(d)和图3-7(e)所示。最后,对整个系统结构进行集成测试,如图3-7(f)所示。

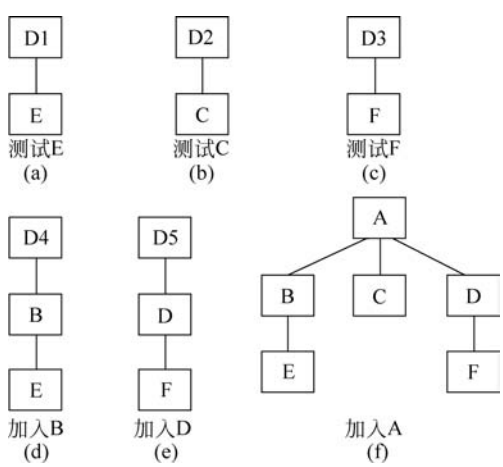

图3-7 自底向上增值测试方式实例

(3) 混合增值测试方式(Modified Top-down Integration)。自顶向下增值的方式和自底向上增值的方式各有优缺点。

自顶向下增值方式的缺点是需要建立桩模块。要使桩模块能够模拟实际子模块的功能十分困难,同时涉及复杂算法。真正输入/输出的模块一般在底层,是最容易出现问题

的模块，并且直到组装和测试的后期才会遇到这些模块，一旦发现问题，将会导致过多的回归测试。

自顶向下增值方式的优点是能够较早地发现在主要控制方面的问题。

自底向上增值方式的缺点是“程序一直未能作为一个实体存在，直到最后一个模块加上去后才形成一个实体”，即在自底向上组装和测试的过程中，对主要的控制直到最后才接触到。

自底向上增值方式的优点是不需要桩模块，建立驱动模块一般比建立桩模块容易，同时由于涉及复杂算法和真正输入/输出的模块最先得到组装和测试，可以把最容易出现问题的部分在早期解决。此外，自底向上增值的方式可以实施多个模块的并行测试。

有鉴于此，通常是把自顶向下增值测试方式和自底向上增值测试方式结合起来进行组装和测试。

改进的自顶向下增值测试：基本思想是强化对输入/输出模块和引入新算法模块的测试，并自底向上组装成为功能相当完整且相对独立的子系统，然后由主模块开始自顶向下进行增值测试。

自底向上—自顶向下的增值测试(混合法)：首先对含读操作的子系统自底向上直至根结点模块进行组装和测试，然后对含写操作的子系统做自顶向下的组装与测试。

回归测试：这种方式采取自顶向下的方式测试被修改的模块及其子模块，然后将这一部分视为子系统，再自底向上测试，以检查该子系统与其上级模块的接口是否适配。

3）一次性集成测试方式与增值式集成测试方式的比较

(1) 增值式集成测试方式需要编写的软件较多，工作量较大，花费的时间较多。一次性集成测试方式的工作量较小。

(2) 增值式集成测试方式发现问题的时间比一次性集成测试方式发现问题的时间早。

(3) 增值式集成测试方式比一次性集成测试方式更容易判断出现问题的原因，因为新出现的问题往往和最后加进来的模块有关。

(4) 增值式集成测试方式测试地更加彻底。

(5) 一次性集成测试方式可以多个模块并行测试。

这两种模式在测试时各有利弊，在时间条件允许的情况下增值式集成测试方式有一定的优势。

4. 集成测试的组织和实施

集成测试是一种正规测试过程，必须精心计划，并与单元测试的完成时间协调起来。在制订测试计划时，应考虑以下因素。

(1) 采用何种系统组装方法来进行组装测试。

(2) 组装测试过程中连接各个模块的顺序。

(3) 模块代码编制和测试进度是否与组装测试的顺序一致。

(4) 测试过程中是否需要专门的硬件设备。

在确定好上述问题之后，就可以列出各个模块的编制、测试计划表，标明每个模块单元测试完成的日期、首次集成测试的日期、集成测试全部完成的日期，以及需要的测试用

例和所期望的测试结果。

在缺少软件测试所需要的硬件设备时,应检查该硬件的交付日期是否与集成测试计划一致。例如,若测试需要数字化仪和绘图仪,则相应的测试应安排在这些设备能够投入使用之时,并需要为硬件的安装和交付使用预留一段时间。此外,在测试计划中需要考虑测试所需软件(驱动模块、桩模块、测试用例生成程序等)的准备情况。

5. 集成测试完成的标志

判定集成测试过程是否完成,可按以下几个方面检查。

(1) 是否成功地执行了测试计划中规定的所有集成测试。

(2) 是否修正了测试时所发现的错误。

(3) 测试结果是否通过了专门小组的评审。

集成测试应由专门的测试小组来进行,测试小组由有经验的系统设计人员和程序员组成。整个测试活动要在评审人员出席的情况下进行。在完成预定的组装测试工作之后,测试小组应负责对测试结果进行整理、分析,形成测试报告。测试报告中要记录实际的测试结果、在测试中发现的问题、解决这些问题的方法以及解决之后再次测试的结果。此外,测试报告还应提出目前不能解决、需要管理人员和开发人员注意的一些问题,提供测试评审和最终决策,提出处理意见。集成测试需要提交的文档有:集成测试计划、集成测试规格说明、集成测试分析报告。

6. 采用集成测试的原因

所有的软件项目都必须经过系统集成这个阶段。不管采用什么开发模式,具体的开发工作总是从一个一个的软件单元开始,软件单元只有经过集成才能形成一个有机的整体。具体的集成过程可能是显性的也可能是隐性的。在软件单元组装过程中通常会出现一些常见问题。集成测试需要花费的时间远超过单元测试,直接从单元测试过渡到系统测试是非常危险的做法,这可能使整个软件开发项目所耗费的时间成倍地增加。集成测试的必要性还在于一些模块虽然能够单独工作,但并不能保证这些模块连接起来也能正常工作。程序在某些局部反映不出来的问题,有可能会在全局上暴露出来,影响软件功能的实现。

3.4 确认测试

1. 确认测试的定义

集成测试完成以后,分散开发的模块被连接起来,构成完整的程序。其中,各模块之间接口存在的问题都已消除。于是,测试工作进入确认测试(Validation Testing)阶段。

什么是确认测试?说法众多,其中最简明、最严格的解释是检验所开发的软件是否能按用户提出的要求运行。若能达到这一要求,则认为开发的软件是合格的。因而有的软件开发部门把确认测试称为合格性测试(Qualification Testing)。这里所说的客户要求通常指的是在软件规格说明书中确定的软件功能和技术指标,或是专门为测试所规定的确认准则。在确认测试阶段需要做的工作如图3-8所示。首先要进行有效性测试以及软件配置审查,然后进行验收测试和安装测试,在通过了专家鉴定之后,才能成为可交付的软件。