第5章 单元测试

一、单元概述

本单元针对课程项目如何进行单元测试而展开。通过本章的学习，读者能够掌握单元测试的定义、内容，能够应用所学的测试用例设计方法进行单元测试设计及执行，能够运用适合的方法对课程项目进行单元测试。

二、教学重点与难点

重点：

(1)单元测试的内容；

(2)单元测试的流程；

(3)单元测试的方法。

难点：

单元测试方法的应用。

5.1 单元测试介绍

5.1.1 单元测试的定义

所谓“单元”是指：

(1)具有明确的功能；

(2)具有明确的规格定义；

(3)具有与其他部分明确的接口定义；

(4)能够与程序的其他部分清晰的进行区分。

其实程序员每天都在做单元测试。编写了一个函数，除了极简单的外，总是要执行，看看功能是否正常，有时还要输出数据，比如弹出信息窗口，这也是单元测试，可以把这种单元测试称为临时单元测试。只进行了临时单元测试的软件，针对代码的测试很不完整，代码覆盖率较低，未覆盖的代码可能遗留大量的细小的错误，这些错误还会互相影响，当 BUG 暴露出来的时候难于调试，大幅度提高后期测试和维护成本，同时降低了开发商的竞争力。因此，进行充分的单元测试是提高软件质量、降低开发成本的必由之路。

要进行充分的单元测试，程序员应专门编写测试代码，并与产品代码隔离。具体的办法

是为产品工程建立对应的测试工程，为每个类建立对应的测试类，为每个函数建立测试函数。

关于单元测试的几个关键问题如下：

(1)单元测试的定义

单元测试又称模块测试，是最小单位的测试，其依据是详细设计规格说明书，对模块内所有重要的控制路径设计测试用例，以便发现模块内部的错误。单元测试多采用白盒测试技术，系统内多个模块可以并行地进行单元测试。

(2)单元测试的对象

一般认为，在结构化程序时代，单元测试所说的单元是指函数，在面向对象编程中，单元测试的单元一般是指类。但从实践来看，以类作为测试单位，复杂度高，可操作性较差，仍然主张以类中的方法作为单元测试的测试单位，但可以用一个测试类来组织某个类的所有测试函数。单元测试不应过分强调面向对象，因为局部代码依然是结构化的。单元测试的工作量较大，简单实用高效才是硬道理。

(3)单元测试的时间

单元测试当然是越早越好，通常在编码阶段进行。在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，就可以开始进行单元测试的测试用例设计。XP(极限编程)开发理论要求测试驱动开发 TDD(Test-Driven Development)，先编写测试代码，再进行开发。在实际的工作中，不必过分强调开发和测试的顺序，重要的是效果。一般是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。

(4)单元测试的人员

绝大部分情况下，由开发人员做单元测试的设计和执行。如果单元测试的需求非常清晰，开发人员之外的人都可以轻易掌握，那么单元测试可以由独立的测试人员完成。但是大部分情况下很难做到这一点，因此就要求开发人员在编写测试用例的时候绝不能假定任何函数的实现，而应该完全按照它应该有的需求来做。

5.1.2　单元测试的重要性

单元测试是软件测试的基础，因此单元测试的效果会直接影响到软件的后期测试，最终在很大程度上影响到产品的质量。

1.时间方面

如果认真的做好了单元测试，在系统集成联调时非常顺利，因此会节约很多时间，反之，那些由于因为时间原因不做单元测试或随便做做的则在集成时总会遇到那些本应该在单元测试就能发现的问题，而这种问题在集成时遇到往往很难让开发人员预料到，最后在苦苦寻觅中才发现这是个很低级的错误而在悔恨自己时已经浪费了很多时间，这种时间上的浪费一点都不值得，正所谓得不偿失。

2.测试效果

根据以往的测试经验来看，单元测试的效果是非常明显的，首先它是测试阶段的基础，做好了单元测试，在做后期的集成测试和系统测试时就很顺利。其次在单元测试过程中能发现一些很深层次的问题，同时还会发现一些很容易发现而在集成测试和系统测试很难发现的问题。再次单元测试关注的范围也特殊，它不仅仅是证明这些代码做了什么，最重要的是代码是如何做的，是否做了它该做的事情而没有做不该做的事情。

3.测试成本

在单元测试时某些问题很容易发现，如果在后期的测试中发现问题所花的成本将成倍数上升。比如在单元测试时发现1个问题需要1个小时，则在集成测试时发现该问题需要2个小时，在系统测试时发现则需要3个小时，同理还有定位问题和解决问题的费用也是成倍数上升的，这就是我们要尽可能早的排除尽可能多的bug来减少后期成本的原因之一。

4.产品质量

单元测试的好与坏直接影响到产品的质量，可能就是由于代码中的某一个小错误就导致了整个产品的质量降低，或者导致更严重的后果，如果我们做好了单元测试，这种情况是可以完全避免的。

5.2 单元测试的内容与方法

5.2.1 单元测试的内容

单元测试的对象是软件设计的最小单位——模块或函数，单元测试的依据是详细设计描述。测试者要根据详细设计说明书和源程序清单，了解模块的I/O条件和模块的逻辑结构。

主要采用白盒测试的测试用例，辅之以黑盒测试的测试用例，使之对任何合理和不合理的输入都能鉴别和响应。要求对所有的局部和全局的数据结构、外部接口和程序代码的关键部分进行桌面检查和代码审查。在单元测试中，需要对下面5个方面的内容进行测试，如图5.1所示，也是测试用例的基础：

(1)模块接口测试；

(2)模块局部数据结构测试；

(3)模块边界条件测试；

(4)模块中所有独立执行通路测试；

(5)模块的各条错误处理通路测试。

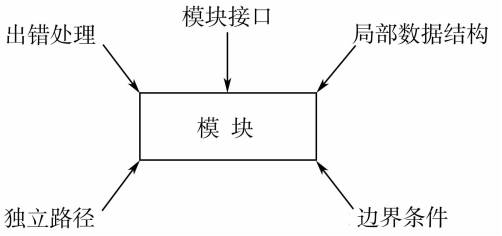

图5.1 单元测试的内容

下面逐个介绍具体的测试内容：

(1)模块接口测试是单元测试的基础。只有在数据能正确流入、流出模块的前提下，其他测试才有意义。测试接口正确与否应该考虑下列因素：

①输入的实际参数与形式参数的个数是否相同；

②输入的实际参数与形式参数的属性是否匹配；

③输入的实际参数与形式参数的量纲是否一致；

④调用其他模块时所给实际参数的个数是否与被调模块的形参个数相同；

⑤调用其他模块时所给实际参数的属性是否与被调模块的形参属性匹配；

⑥调用其他模块时所给实际参数的量纲是否与被调模块的形参量纲一致；

⑦调用预定义函数时所用参数的个数、属性和次序是否正确；

⑧是否存在与当前入口点无关的参数引用；

⑨是否修改了只读型参数；

⑩对全程变量的定义各模块是否一致；

⑪是否把某些约束作为参数传递。

如果模块内包括外部输入输出，还应该考虑下列因素：

①文件属性是否正确；

②OPEN/CLOSE 语句是否正确；

③格式说明与输入输出语句是否匹配；

④缓冲区大小与记录长度是否匹配；

⑤文件使用前是否已经打开；

⑥是否处理了文件尾；

⑦是否处理了输入/输出错误；

⑧输出信息中是否有文字性错误。

(2)局部数据结构测试是为了保证临时存储在模块内的数据在程序执行过程中完整、正确。局部数据结构往往是错误的根源，应仔细设计测试用例，力求发现下面几类错误：

①不合适或不相容的类型说明；

②变量无初值；

③变量初始化或缺省值有错；

④变量名拼写错误或书写错误；

⑤出现上溢、下溢和地址异常；

⑥使用尚未赋值或尚未初始化的变量。

除了局部数据结构外，如果可能，单元测试时还应该查清全局数据(例如 FORTRAN 的公用区)对模块的影响。

(3)路径测试应对每一条独立执行路径进行测试，单元测试的基本任务是保证模块中每条语句至少执行一次。基本路径测试和循环测试是最常用且最有效的测试技术，设计测试用例查找由于错误的计算、不正确的比较或不正常的控制流而导致的错误。计算中常见的错误包括：

①常见的不正确的计算有：运算的优先次序不正确或误解了运算的优先次序；运算的方式错误(运算的对象彼此在类型上不相容)；算法错误；初始化不正确；运算精度不够；表达式的符号表示不正确等。

②常见的比较和控制流错误有：不同数据类型的比较；不正确的逻辑运算符或优先次序；浮点数的比较不等；关系表达式中不正确的变量和比较符；多循环或少循环一次；错误的

或不可能的循环终止条件；当遇到发散的迭代时不能终止循环；不适当地修改了循环变量等。

(4)错误处理测试，比较完善的模块设计要求能预见出错的条件，并设置适当的出错处理对策，以便在程序出错时，能对出错程序重新做安排，保证其逻辑上的正确性。这种出错处理也是模块功能的一部分。测试应着重检查下列问题：

①输出的出错信息难以理解；

②记录的错误与实际遇到的错误不相符；

③在程序自定义的出错处理运行之前，系统已介入；

④异常处理不当；

⑤错误陈述中未能提供足够的定位出错信息；

⑥如果出错情况不予考虑，那么检查恢复正常后模块可否正常工作。

(5)边界测试是单元测试中最后，也是最重要的一项任务。众所周知，软件经常在边界上失效，采用边界值分析技术，针对边界值设计测试用例，很有可能发现新的错误。设计测试用例检查：

①在 n 次循环的第 0 次、1 次、2 次、n－1 次、n 次、n＋1 次是否有错误；

②取最大、最小值时是否有错误；

③取空值、空串、空引用；

④达到或超出最大长度、最大值；

⑤刚好等于、大于、小于确定的比较值时是否出现错误。

5.2.2 单元测试的方法

以下是一些单元测试案例的常见设计方法，通过对这些方法的综合运用，可以帮助我们发现上述这些错误。

1.规格导出法

规格导出法是根据相关的规格说明来设计测试用例，每一个测试用例用来检验一个或多个规格陈述的语句。一个比较实际的办法是按照规格陈述的语句顺序来为被测单元设计测试用例。这种测试用例的设计可以保证在规格说明中所有的要求在测试案例中都能得到体现，但是它只是一种正向测试的思路，需要其他的测试用例的补充才能达成测试的完整性。

2.等价类划分法

等价类划分是一种正式的测试用例设计方法，它基于被测单元的输入、输出所做的划分，对每一个划分中的所有输入被测单元都有相同(等价)的反应。例如对一个范围是 0～100 的整数输入来说，2、38、66 应该都具有相同的效力，而－1、120 也有相同的效力。等价类划分法就是针对每一个等价类设计至少一个测试案例来确保被测程序单元的处理是完整的。等价类划分的设计方法也属于正向测试的技术。

3.边界值分析法

边界值分析法使用与等价类划分法相同的划分，只是边界值分析假定错误更多地存在于两个划分的边界上，相应地为边界上及两侧的情况设计测试用例。

4.状态转移测试

对于那些以状态机作为模型或者设计为状态机的软件,状态转移测试是合适的。状态转移测试法的测试案例涵盖能导致状态迁移的事件,用以测试状态之间的转换是否正确。用这种方法可以测试逆向的测试用例,如状态和事件的非法组合。

5.分支测试法

在分支测试中,根据单元中控制流分支或者判断点来设计测试用例。这通常用于达到一定的测试覆盖率。在单元测试中,如果使用黑盒测试技术,那么需要去猜测存在哪些逻辑分支并相应为这些分支的执行准备测试用例,如果使用白盒测试技术,那么则需要根据该程序单元中的控制流设计测试案例,完成分支覆盖的要求。

6.条件测试法

条件测试法中包涵了很多测试案例设计技术,它们都致力于弥补在遇到复杂逻辑条件的时候分支测试的弱点。条件测试的目标是测试在每个逻辑条件的单个成份及它们组合的情况下程序都是正确的。在考虑各个逻辑条件的组合的时候,决策表是一种有用的工具。在条件测试法中,需要设计足够的测试案例,确保每种逻辑条件的组合都能够被测试到。

7.数据定义——使用测试法

数据定义是指数据被赋值的地方,数据使用是指数据项被读取或者使用的地方。使用这种方法设计测试案例时,主要考虑用案例来驱动数据被定义到被使用的路径。这种方法主要用于检查数据的初始化和处理的正确性,也可以在静态检查中使用。

8.内部边界值测试法

这种方法与边界值分析法类似,但是它偏重的是白盒测试技术,也就是说从程序单元的规格说明中导出等价类和边界值。除了外部可见的数据之外,程序的内部的数据也存在等价类和边界值,它们只能通过对程序单元的设计规格说明进行分析而得到。内部边界值测试法一般只作为测试案例设计的补充方法,与其他方法结合使用。

9.错误猜测法

错误猜测是基于经验和其他一些测试技术的。在经验的基础上,测试设计者猜测错误的类型及在特定的软件中错误发生的位置,并设计测试用例去发现它们。例如,如果所有的资源需要动态申请,那么我们就需要判断是否所有的资源都被正确释放了。一个发现错误的好地方就是资源释放的地方。对一个有经验的工程师,错误猜测法可能是最好的设计测试案例的方法,因为它可能发现别的设计方法所遗漏的错误。为了最大限度的利用有效的经验并逐步丰富测试用例的设计技术,建立一个错误类型的列表是一个好方法,这个列表可以帮助工程师猜测程序单元中的错误会在哪里。这个列表需要通过在实践中不断的维护和扩充来帮助达成错误猜测的有效性。

10.循环测试法

重点检查循环的条件-判断部分以及边界条件。测试循环是一种特殊的路径测试,因为循环比其他语句都复杂一些。循环中错误的发生机会比其他代码构成部分多。对于单层循环要采用边界值测试法,对于嵌套循环应用如下方法设计测试用例:

(1)把外循环设置为最小值,并运行内循环所有可能的情况;

(2)把内循环设置为最小值,并运行外循环所有可能的情况;

(3)把所有的循环变量都设置为最小值运行;

(4)把所有的循环变量都设置为最大值运行;

(5)把外循环设置为最大值,并运行内循环所有可能的情况;

(6)把内循环设置为最大值,并运行外循环所有可能的情况。

5.3 单元测试的过程

单元测试是对软件基本组成单元进行的测试,单元测试的侧重点在于发现程序设计或者实现中的逻辑错误。它分为计划、设计、执行和评估四个步骤。各步骤的定义如下:

(1)计划单元测试:确定测试需求,制订测试策略,确定测试所用资源,创建测试任务的时间表。

(2)设计单元测试:根据单元测试计划设计单元测试模型,制订测试方案,确认测试过程,制订具体的测试用例,创建可重用的测试脚本。

(3)执行单元测试:根据单元测试的方案、用例对软件单元进行测试,验证测试结果,并记录测试过程中出现的缺陷。

(4)评估单元测试:对单元测试的结果进行评估,主要从需求覆盖和代码覆盖的角度进行测试完备性的评估。

5.3.1 计划单元测试

1.确定测试需求

单元测试其实难在测试策略上,对于一个上百万行代码量的软件系统,要完成所有单元模块的测试对于很多软件企业来说几乎是不可能的。所以,在测试过程中由于时间或资源的原因可能会使测试处于紧张的局面,如果制定一个好的,有效的测试策略是至关重要的,也是能够有效地进行单元测试活动的途径。策略来源于为各模块制定测试优先级,其优先级的划分依据如下:

(1)哪些是重点模块?

(2)哪些程序是最复杂、最容易出错的?

(3)哪些程序是相对独立,应当提前测试的?

(4)哪些程序最容易扩散错误?

(5)哪些程序是开发者最没有信心的?

(6)8020:80%的缺陷聚集在20%的模块中,经常出错的模块改错后还会经常出错?

(7)哪些是底层模块?

(8)哪些是使用频率最多的模块?

单元测试的流程如图5.2所示,需要经过单元测试计划、单元测试用例设计、评审以及执行测试、bug跟踪、提交报告等过程。

2.确定测试策略

一旦明确单元测试的重点,接下来就需要进一步确认应用什么样的测试方法。具体的

方法在前面已经介绍了，这里给出一个综合的策略：

首先，根据《需求规格说明书》和《概要设计说明书》、《详细设计说明书》，应用场景法、等价类划分法、规格导出法、状态转移法等检查程序是否正确的实现了功能。

其次，采用静态测试方法，如代码审查、走查、桌面检查，重点在于检查代码是否符合编码规范，模块接口是否正确。

然后，应用条件测试法、分支测试法和循环测试法等测试程序路径，实现语句覆盖、判定覆盖、条件覆盖。

最后，应用边界值分析、错误猜测、健壮性分析等方法重点考察边界、异常、错误处理是否符合要求。

3.单元测试的输入

软件需求规格说明书，软件详细设计说明书，软件编码与单元测试工作任务书，软件集成测试计划，软件集成测试方案，用户文档。

4.单元测试的输出

单元测试计划，单元测试方案，需求跟踪说明书或需求跟踪记录，代码静态检查记录，正规检视报告，问题记录，问题跟踪和解决记录，软件代码开发版本。

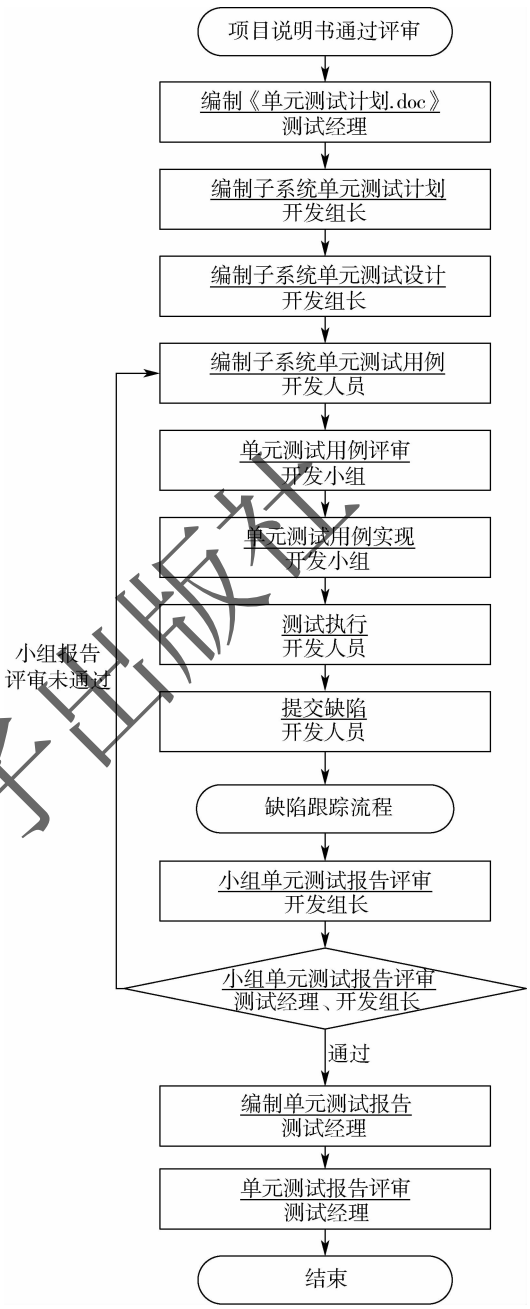

图 5.2　单元测试的流程

5.3.2　设计单元测试

1.单元测试的模型

在单元测试时，如果模块不是独立的程序，需要辅助测试模块，有两种辅助模块，如图 5.3所示。

(1)驱动模块(Driver)：所测模块的主程序。它接收测试数据，把这些数据传递给所测试模块，最后再输出测试结果。当被测试模块能完成一定功能时，也可以不要驱动模块。

(2)桩模块(Stub)：用来代替所测模块调用的子模块。被测试模块、驱动模块和桩模块共同构成了一个测试模型。

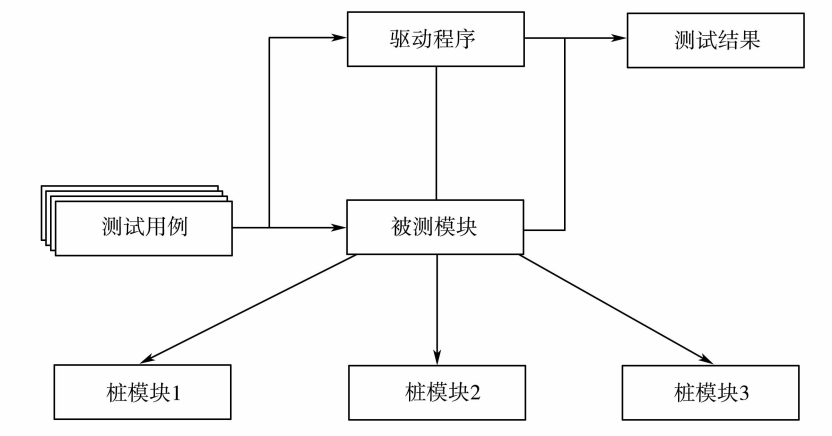

图 5.3 单元测试的模型

2.单元测试的方案

在制定测试计划的阶段已经明确了此次单元测试的主体策略，本阶段需要具体到每个模块的测试角度以及测试方法的选择，通常包括常规的测试和特定的测试两种情况，如下：

常规的用例设计方法：规格导出法，等价类划分法，边界值分析法，状态转移测试法，分支测试法，条件测试法，数据定义－使用测试法，内部边界值测试法，错误猜测法，循环测试法。

特定的用例设计方法：

(1)声明测试：检查模块中的所有变量是否被声明。经验表明，大量重要的错误都是由于变量没有被声明或没有被正确的声明而引起的。

(2)路径测试：要求模块中所有可能的路径都被执行一遍，属逻辑覆盖测试。

(3)基本路径测试：由于实际中，一个模块中的路径可能非常多，由于时间和资源有限，不可能一一测试到。这就需要把测试所有可能路径的目标减少到测试足够多的路径，以获得对模块的信心。要测试的最小路径集就是基本测试路径集。基本测试路径集要保证：每个确定语句的每一个方向要测试到，每条语句最少执行一次。

(4)循环测试：重点检查循环的条件－判断部分以及边界条件。测试循环是一种特殊的路径测试，循环中错误的发生机会比其他代码构成部分多，在前面单元测试的方法中已经进行了具体的介绍。

(5)边界值测试：确定代码在任何边界情况下都不会出差错。重点检查小于、等于和大于边界条件的情况。边界值测试是指专门设计用来测试当条件语句中引用的值处在边界或边界附近时系统反映的测试。

(6)接口测试：检查模块的数据流(输入、输出)是否正确。检查输入的参数和声明的自变量的个数，数据类型和输入顺序是否一致。检查全局变量是否被正确的定义和使用等。

(7)确认测试：是否接受有效输入数据(操作)，拒绝无效数据(操作)。

(8)事务测试：输入能否正确输出，错误是否处理。

3.测试用例的设计

(1)测试用例的设计原则：一个好的测试用例在于能够发现至今没有发现的错误；测试用例应由测试输入数据和与之对应的预期输出结果这两部分组成；在测试用例设计时，应当

包含合理的输入条件和不合理的输入条件；为系统运行起来而设计测试用例；为正向测试而设计测试用例；为逆向测试而设计测试用例；为满足特殊需求而设计测试用例；为代码覆盖而设计测试用例。

图 5.4　登录界面

(2)测试用例的规范，表 5.1 是图 5.4 登录功能的测试用例的一个示例，仅供参考。通常测试用例应该包括如下信息：用例运行前置条件，被测模块/单元所需环境(全局变量赋值或初始化实体)，启动测试驱动，设置桩，调用被测模块，设置预期输出条件判断，恢复环境(包括清除桩)。

表 5.1　登录界面测试用例

项目名称			程序版本	
测试环境	硬件环境 服务器端：IBM 小型机 客户端：2 台 PC(CPU：P4.2.4G RAM：256M)			
	软件环境 服务器端：操作系统——Linux 9.0；数据库——Oracle 9i；Web 服务器——Web sphere4.0 客户端：操作系统——windows 2000 pro；浏览器——IE 5.0			
	网络环境 公司内部的以太网，与服务器的连接速率为 100M，与客户端的连接速率为 10、100M			
编制人			编制时间	
功能模块	用户登录			
功能特性	用户身份验证			
测试目的	验证是否输入合法的信息			
预置条件	在后台添加 1 个前台用户，用户名为 user，密码为 a1；进入网站前台首页			
参考信息	需求说明中关于“登录”的说明			
用例编号	测试步骤	输入数据	预期结果	测试结果
DL001	输入用户名和密码 按“提交”按钮	用户名＝user 密码＝a1		
DL002				
DL003				
DL004				
DL005				

5.3.3　执行单元测试

执行单元测试应遵循以下步骤：

(1)设置测试环境，以确保所有必需的元素(硬件、软件、工具、数据等)已得到实施，并且都处于测试环境中。

(2)将测试环境初始化,以确保所有构件都处于正确的初始状态。

(3)执行测试过程。需要注意的是测试过程的执行将随着具体情况而变化:测试方式是自动还是人工,以及必需的测试构件是作为驱动程序还是桩模块。自动测试的测试脚本在执行实施测试步骤的过程中创建,而人工测试则是在“构建测试过程”活动中制定的结构化测试过程。

单元测试何时要终止呢?测试执行在出现以下两个条件之一时结束或终止:

①正常:所有测试过程(或脚本)按预期方式执行。如果测试正常终止,则继续执行“核实测试结果”活动,目的在于确定测试结果是否可靠。

②异常或提前结束:测试过程(或脚本)没有按预期方式执行或没有完全执行。当测试异常终止时,测试结果可能不可靠。需要确定和纠正测试终止的原因,并在执行其他测试活动之前重新执行此测试。如果测试异常终止,则继续执行“恢复暂停的测试活动”,其目的在于确定测试是否成功完成,是否符合预期目标。

针对测试结果表明的测试过程或测试工作中存在的缺陷,确定合适的纠正措施,及时的补充测试用例以及更新测试用例文档。测试完成后,应当复审测试结果以确保测试结果可靠,确保所报告的故障、警告或意外结果不是外部影响(例如,不正确的设置或数据等)造成的。

如果所报告的故障是在测试工作中确定的错误导致的,或者是测试环境的问题造成的,则应当采取适当的纠正措施进行纠正,然后重新执行测试。

如果测试结果表明故障确实是由测试目标引起的,则完成“执行测试活动”后,下一步的活动是评估测试。

5.3.4 评估单元测试

单元测试完成以后,需要对单元测试的执行效果进行评估,主要从以下几方面进行:

(1)测试完备性评估。主要检查测试过程中是否已经执行了所有的测试用例,对新增的测试用例是否已及时更新测试方案等。

(2)代码覆盖率评估。主要是根据代码覆盖率工具提供的语句覆盖情况报告,检查是否达到方案中的要求,大多数情况下,要求语句覆盖达到100%。但很多情况下,第一轮测试用例执行完成后是很难达到的,这时在评估过程中要对覆盖率进行分析,主要从以下方面来考虑:不可能的路径或条件,不可达的或冗余的代码,不充分的测试用例。

(3)从覆盖的角度看。测试应该做到以下覆盖:功能覆盖,输入域覆盖,输出域覆盖,函数交互覆盖,代码执行覆盖。

大多数有效的测试用例都来自于分析,而不是仅仅为了达到测试覆盖率目标而草率设计测试用例。测试覆盖并不是最终的目的,它只是评价测试的一种方式,为测试提供指导和依据。

5.4 教学管理平台的单元测试

【项目5-1】针对教学管理平台的专业信息模块进行单元测试。

【项目构思】专业信息模块包含新建、修改、浏览和删除四部分，该模块能够实现专业名称、专业负责人、开设时间、目前状态和专业层次等信息的管理，针对此需求进行单元测试。

【项目设计】

- 确定测试策略：包含测试的方法和重点；
- 设计测试用例，必须体现预期结果；
- 执行单元测试；
- 撰写单元测试总结。

【项目实施】

步骤 1　程序界面如图 5.5～图 5.7 所示。

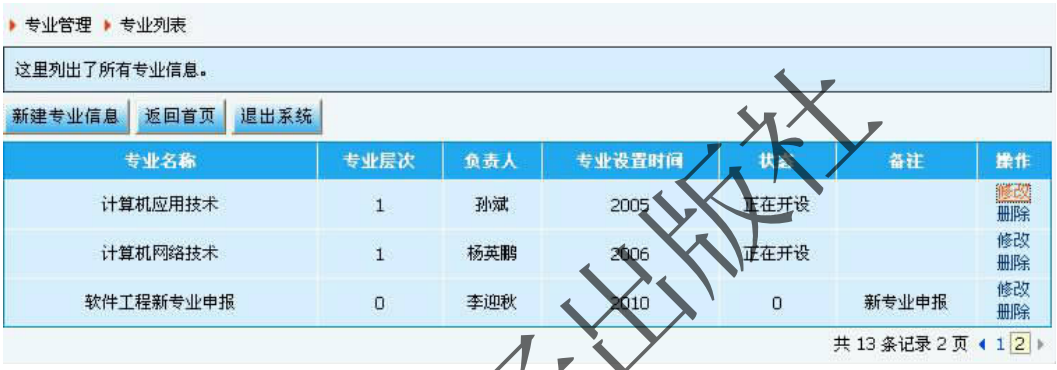

图 5.5　浏览专业信息页面

图 5.6　修改专业信息页面

图 5.7　新建专业信息

步骤 2　其核心的方法如下：

```
//添加课程信息
    public String addMajor()throws Exception {
        try {
    getServMgr().getMajorService().addMajor(major_name,major_level,teacher_id,major_set_
date,major_status,major_memo);
            addMessage("专业创建成功");
            addRedirURL("返回","major! majorList.action");
        } catch (DataIntegrityViolationException e) {
            setResult( ERROR );
```

```
            addMessage("专业创建失败");
            log.debug(e.toString());
            addRedirURL("返回","@back");
        }
        return EXECUTE_RESULT;
    }

    //修改专业信息
    public String editMajor()throws Exception {
        try {
    getServMgr().getMajorService().updateMajor(major_id,major_name,major_level,teacher_id,
major_set_date,major_status,major_memo);
            addMessage("更改课程信息成功");
            addRedirURL("返回","major! majorList.action");
        } catch (DataIntegrityViolationException e) {
            setResult(ERROR);
            addMessage("更改课程信息失败");
            addMessage(e.toString());
            addRedirURL("返回","@back");
        }
        return EXECUTE_RESULT;
        }
        //删除专业信息
        public String delete()throws Exception {
            try {
                getServMgr().getMajorService().delMajor(major_id);
                addMessage("专业删除成功");
                addRedirURL("返回","major! majorList.action");
            } catch (DataIntegrityViolationException e) {
                setResult(ERROR);
                addMessage("专业删除失败");
                addRedirURL("返回","@back");
            }
            return EXECUTE_RESULT;
            }
```

步骤3　针对每个方法测试其接口、局部数据结构、边界条件、独立执行通路和错误处理通路，分别设计测试用例。

步骤4　执行测试并记录结果。

步骤5　提交单元测试总结。

思考题

1. 编写三角形问题的程序，学习小组实施代码评审，并提交评审结论。

2. 编写 NextDate 问题的程序，编写驱动模块和所需的桩模块，应用恰当的测试用例设计方法设计测试用例并执行测试。