目　录

第 1 章　软件测试概述

1. 简述软件测试的意义。

解：随着计算机技术的迅速发展和广泛深入的应用，软件质量问题已成为开发和使用软件人员关注的焦点。而由于软件本身的特性，软件中的错误是不开避免的。不断改进的开发技术和工具只能减少错误的发生，但是却不可能完全避免错误。因此为了保证软件质量，必须对软件进行测试。软件测试是软件开发中必不可少的环节，是最有效的排除和防治软件缺陷的手段，是保证软件质量、提高软件可靠性的最重要手段。

2. 什么是软件缺陷？它的表现形式有哪些？

解：从产品内部看，软件缺陷是软件产品开发或维护过程中所存在的错误、毛病等各种问题；从外部看，软件缺陷是系统所需实现的某种功能的失效或违背。

它的表现形式主要有以下几种：（1）软件未达到产品说明书中已经标明的功能；（2）软件出现了产品说明书中指明不会出现的错误；（3）软件未达到产品说明书中虽未指出但应当达到的目标；（4）软件功能超出了产品说明书中指出的范围；（5）软件测试人员认为软件难以理解、不易使用，或者最终用户认为该软件使用效果不良。

3. 简单分析软件缺陷产生的原因，其中那个阶段引入的缺陷最多，修复成本又最低？

解：软件缺陷产生的主要原因有：需求规格说明错误；设计错误；程序代码有误；其他。其中在需求分析阶段引入的缺陷最多，修复的成本又最低。

4. 当用户登录某网站购物完毕并退出后，忽然想查查购物时付账的总金额，于是按了浏览器左上角的“退回”按钮，就又回到了退出前的网页，你认为该购物软件有缺陷吗？如果有，属于哪一类？

解：有缺陷。其所属类别与软件产品说明书的要求有关。

5. 什么是软件测试？简述其目的与原则。

解：软件测试是为了尽快尽早地发现在软件产品中所存在的各种软件缺陷而展开的贯穿整个软件开发生命周期，对软件产品（包括阶段性产品）进行验证和确认的活动过程。

测试目的：（1）证明：获取系统在可接受风险范围内可用的信心；尝试在非正常情况和条件下的功能和特性；保证一个工作产品是完整的并且可用或可被集成。（2）检测：发现缺陷、错误和系统不足；定义系统的能力和局限性；提供组件、工作产品和系统的质量信息。（3）预防：澄清系统的规格和性能；提供预防或减少可能制造错误的信息；在过程中尽早检测错误；确认问题和风险，并且提前确认解决这些问题和风险的途径。

测试过程中应注意和遵循的原则：（1）测试不是为了证明程序的正确性，而是为了证明程序不能工作。（2）测试应当有重点。（3）事先定义好产品的质量标准。（4）软件项目一启动，软件测试也就开始，而不是等到程序写完才开始进行测试。（5）穷举测试是不可能的。（6）第三方进行测试会更客观，更有效。（7）软件测试计划是做好软件测试工作的前提。

（8）测试用例是设计出来的，不是写出来的。（9）对发现错误较多的程序段，应进行更深入的测试。（10）重视文档，妥善保存一切测试过程文档。

6. 件测试阶段是如何划分的？

解：软件测试的阶段划分为：规格说明书审查；系统和程序设计审查；单元测试；集成测试；确认测试；系统测试；验收测试。

7. 简述软件开发的几个模式，并说明每种模式对软件测试的影响。

解：大棒模式简单，计划、进度安排和正规开发过程几乎没有，其开发过程是非工程化的。大棒模式的软件测试通常在开发任务完成后进行，很难回头修复存在的问题，测试工作只是向客户报告软件经过测试后发现的情况。

边写边改模式通常最初只有粗略的想法就进行简单的设计，然后开始较长的反复编写、测试和修复过程，在认为无法更精细地描述软件产品要求时就发布产品。该模式下，软件测试人员将和程序员一起陷入可能是长期的循环往复过程。

瀑布模式将软件生命周期的各项活动规定为按照固定顺序相连的若干个阶段性工作，形如瀑布流水，最终得到软件产品。软件测试在后期展开，使得开发中出现的问题直到开发后期才显露，失去了及早纠正的机会。

快速原型模式首先构造一个功能简单的原型系统，然后通过对原型系统逐步求精，不断扩充完善得到最终的软件系统。原型系统在扩充完善过程中不断被检查、测试和修改。

螺旋模式是瀑布模式与边写边改模式演化结合的形式，并加入了风险评估所建立的软件开发模式，其主要思想是在开始时不必详细定义所有细节，而是从小开始，定义重要功能，尽量实现，接受客户反馈，进入下一阶段并重复上述过程，直到获得最终产品。测试在每个阶段都要进行，并从最初就参与。

8. 简述软件测试过程。

解：软件测试过程主要包括如下 6 个活动：测试计划；测试需求分析；测试设计；测试规程实现；测试执行；总结生成报告。

9. “软件测试能够保证软件的质量”这句话对吗？软件测试和软件质量之间是什么关系？

解：不对。软件测试是保障软件质量的手段之一，但不是唯一手段。测试是产品高质量的必要非充分条件，软件测试不能决定软件质量。

10. 判断以下说法是否正确。

（1）软件测试和软件调试是同一回事。

（2）软件测试是可以无穷尽的。

（3）测试是为了证明软件的正确性。

（4）测试过程中应重视测试的执行，可以轻视测试的设计。

（5）测试不能修复所有的软件故障。

（6）因为测试工作简单，对软件产品影响不大，所以可以把测试作为新员工的一个过渡工作，或安排不合格的开发人员做测试。

解：（1）（2）（3）（4）（6）错误，（5）正确。

11. 简述软件开发进程与测试进程的关系。

解：软件测试是一个贯穿软件开发生命周期的活动，它可以是一个与开发并行的过程，也可以是在开发完成某个阶段任务之后的活动。

第 2 章　软件测试方法与过程

1．对软件测试的复杂性进行归纳分析。

解：软件测试的复杂性在于：无法对程序进行完全的测试；测试无法保证被测程序中无遗留错误；不能修复所有的软件故障。

2．分别解释什么是静态测试、动态测试、黑盒测试、白盒测试、人工测试和自动化测试。

解：所谓静态测试是指不运行被测软件，仅通过分析或检查等其他手段达到检测的目的。

所谓动态测试是指通过运行被测软件，检查运行结果与预期结果的差异，并分析运行效率和健壮性等性能。

黑盒测试是指在对程序进行的功能抽象的基础上，将程序划分成功能单元，然后对每个功能单元生成测试数据进行测试。用这种方法进行测试时，被测程序被当作打不开的黑盒，因而无法了解其内部构造，因此又称为功能测试。

白盒测试又称为结构测试或逻辑驱动测试，它是知道产品内部工作过程，可通过测试来检测产品内部动作是否按照规格说明书的规定正常进行，按照程序内部的结构测试程序，检验程序中的每条通路是否都有能按预定要求正确工作，而不顾它的功能。

广义上，人工测试是人为测试和手工测试的统称。人为测试的主要方法有桌前检查，代码审查和走查。手工测试指的是在测试过程中，按测试计划一步一步执行程序，得出测试结果并进行分析的测试行为。

自动化测试指的是利用测试工具来执行测试，并进行测试结果分析的测试行为。

3．如果没有软件规格说明或需求文档，可以进行动态黑盒测试吗？为什么？

解：不行。因为黑盒测试是基于软件规格说明的测试。

4．在单元测试中，所谓单元是如何划分的？

解：单元测试的对象通常是软件设计的最小逻辑单元，单元的划分在面向过程的结构化程序中一般是函数或子过程，在面向对象的程序中可以是类或类的成员函数。

5．简述单元测试的主要任务。

解：单元测试的主要任务是：模块接口测试；局部数据结构测试；路径测试；错误处理测试；边界测试。

6．如果开发时间紧迫，是否可以跳过单元测试而直接进行集成测试？为什么？

解：不可以。因为没有经过单元测试的模块会遗留大量的缺陷到集成测试阶段，而在集成测试阶段对这些缺陷定位困难，导致后续工作展开困难，修复缺陷成本成指数级增长。

7．什么是驱动模块和桩模块？为下面的函数构造一个驱动模块。

int divide(int a, int b)

```
{
   int c;
   if (b==0) {printf("除数不能为 0"); return 0;}
   c=a/b;
   return c;
}
```

解：驱动模块是用以模拟被测模块的上级模块，它接收测试数据，传送数据给被测模块，启动被测模块，最后输出实测结果。

桩模块用以模拟被测模块工作过程中所调用的子模块。

函数驱动模块：

```
void main( )
{
   int x,y,z;
   scanf(“%d%d”,&x,&y);
   z=divide(x,y);
   printf(“%d”,z);
}
```

8．什么是回归测试？什么时候进行回归测试？

解：回归测试就是重新运行现有测试用例测试原有功能，以便确定变更是否达到了预期的目的，检查变更是否损害了原有的正常功能。每当软件发生变化时就应进行回归测试。

9．集成测试有哪些不同的集成方法？简述不同方法的特点。

解：集成测试通常有一次性集成、自顶向下集成、自底向上集成和混合集成 4 种集成方法。

一次性集成方法需要的测试用例数目少，测试方法简单、易行。但是由于不可避免存在模块间接口、全局数据结构等方面的问题，所以一次运行成功的可能性不大；如果一次集成的模块数量多，集成测试后可能会出现大量的错误，给程序的错误定位与修改带来很大的麻烦；即使集成测试通过，也会遗漏很多错误进入系统测试。

自顶向下集成在测试的过程中，可以较早地验证主要的控制和判断点；一般不需要驱动程序，减少了测试驱动程序开发和维护的费用；可以和开发设计工作一起并行执行集成测试，能够灵活的适应目标环境；容易进行故障隔离和错误定位。但是在测试时需要为每个模块的下层模块提供桩模块，桩模块的开发和维护费用大；桩模块不能反映真实情况，重要数据不能及时回送到上层模块，导致测试不充分；涉及复杂算法和真正 I/O 的底层模块最易出问题，在后期才遇到导致过多的回归测试。

自底向上集成可以尽早的验证底层模块的行为；提高了测试效率；一般不需要桩模块；容易对错误进行定位。但是直到最后一个模块加进去之后才能看到整个系统的框架；驱动模块的设计工作量大；不能及时发现高层模块设计上的错误。

混合集成具有自顶向下和自底向上两种集成策略的优点，但是在被集成之前，中间层不能尽早得到充分的测试。

10．系统测试主要包括哪些内容？

解：系统测试主要包括强度测试、性能测试、恢复测试、安全测试、可靠性测试、安装测试、

容量测试和文档测试。

11．验收测试是由谁完成的？通常包含哪些过程？

解：验收测试是以用户为主的测试，软件开发人员和 QA（质量保证）人员也应参加。通常包含 α 测试和 β 测试过程。

12．分析比较面向对象的软件测试与传统的软件测试的异同。

解：传统的单元测试的对象是软件设计的最小单位——模块。当考虑面向对象软件时，单元的概念发生了变化，此时最小的可测试单位是封装的类或对象，而不再是个体的模块。传统单元测试主要关注模块的算法实现和模块接口间数据的传递，而面向对象的单元测试主要考察封装在一个类中的方法和类的状态行为。

面向对象软件没有层次的控制结构，因此传统的自顶向下和自底向上集成策略就不再适合，它主要有以下两种集成策略： 基于类间协作关系的横向测试；基于类间继承关系的纵向测试。

系统测试一般不考虑内部结构和中间结果，因此面向对象软件系统测试与传统的系统测试差别不大。

面向对象软件测试的整体目标和传统软件测试的目标是一致的，即以最小的工作量发现尽可能多的错误，但是面向对象测试的策略和战术有很大不同。测试的视角扩大到包括复审分析和设计模型，此外，测试的焦点从过程构件(模块)移向了类。

第 3 章　黑盒测试

1．分析黑盒测试方法的特点。

解：黑盒测试又称为功能测试或数据驱动测试，主要针对软件界面、软件功能、外部数据库访问以及软件初始化等方面进行测试。

优点：1）比较简单，不需要了解程序内部的代码及实现；2）与软件的内部实现无关；3）从用户角度出发，能很容易的知道用户会用到哪些功能，会遇到哪些问题；4）基于软件开发文档，所以也能知道软件实现了文档中的哪些功能；5）在做软件自动化测试时较为方便。

缺点：1）不可能覆盖所有的代码，覆盖率较低，大概只能达到总代码量的 30%；2）自动化测试的复用性较低。

2．健壮等价类测试与标准等价类测试的主要区别是什么？

解：主要区别在于健壮等价类测试在标准等价类的基础上还要进行有效取值范围之外的输入（无效输入）的测试。

3．试用等价分类法测试党政管理系统中党员出生年月的输入设计是否符合要求，假设出生年月格式为 yyyymmdd。

解：

输入数据	无效等价类	有效等价类
出生年月日	①8 位数字字符	②有非数字字符 ③少于 8 个数字符 ④多于 8 个数字符
对应数值	⑤在 19090101-19900101 之间	⑥<19090101 ⑦>19900101
月份对应数值	⑧在 1-12 之间	⑨等于"00 " ⑩>12
日期对应值	⑪1,3,5,7,8,10,12 月在 1-31 之间 ⑫4,6,9,11 月在 1-30 之间 ⑬闰年 2 月在 1-29 之间 ⑭非闰年 2 月在 1-28 之间	⑮等于"00 " ⑯>31 ⑰2,4,6,9,11 月等于"31 " ⑱2 月等于"30 " ⑲非闰年 2 月等于"29"

4．找零钱最佳组合：假设商店货品价格(R)皆不大于 100 元（且为整数），若顾客付款在 100 元内(P)，求找给顾客之最少货币个（张）数？（货币面值 50 元(N50)，10 元(N10)，5 元(N5)，1 元(N1)四种。试根据边界值法设计测试用例。

解：1）分析输入的边界情况：

R > 100	0 < R < = 100	R <= 0
P > 100	R<= P <= 100	P < R

2）分析零钱最佳组合的输出情况：

N50 = 1　　N50 = 0

4>N10>= 1　　N10 = 0

N5 = 1　　N5 = 0

4>N1>= 1　　N1 = 0

3）分析规格中每一决策点之情形，以 RR1,RR2,RR3 表示计算要找 50,10,5 元货币数时的剩余金额。

R > 100　　R <= 0

P > 100　　P < R

RR1 >= 50　　RR2 >= 10　　RR3 >= 5

4）根据上述的输入/输出条件组合出可能的情况：

R > 100

R <= 0

0 < R <= 100, P > 100

0 < R <= 100, P < R

0 < R <= 100, R <= P <= 100, RR = 50

0 < R <= 100, R <= P <= 100, RR = 49

0 < R <= 100, R <= P <= 100, RR = 10

0 < R <= 100, R <= P <= 100, RR = 9

0 < R <= 100, R <= P <= 100, RR = 5

0 < R <= 100, R <= P <= 100, RR = 4

0 < R <= 100, R <= P <= 100, RR = 1

0 < R <= 100, R <= P <= 100, RR = 0

5）为满足以上各种情形，测试用例设计如下：

测试用例	货品价格 R	付款金额 P
test1	101	-
test2	0	-
test3	-1	-
test4	100	101
test5	100	99
test6	50	100
test7	51	100
test8	90	100
test9	91	100
test10	95	100
test11	96	100
test12	99	100
test13	100	100

5．试为三角形问题中的直角三角形开发一个决策表和相应的测试用例。注意，会有等腰直角三角形。

解：判断构成的是否为直角三角形的问题的决策表设计如下：

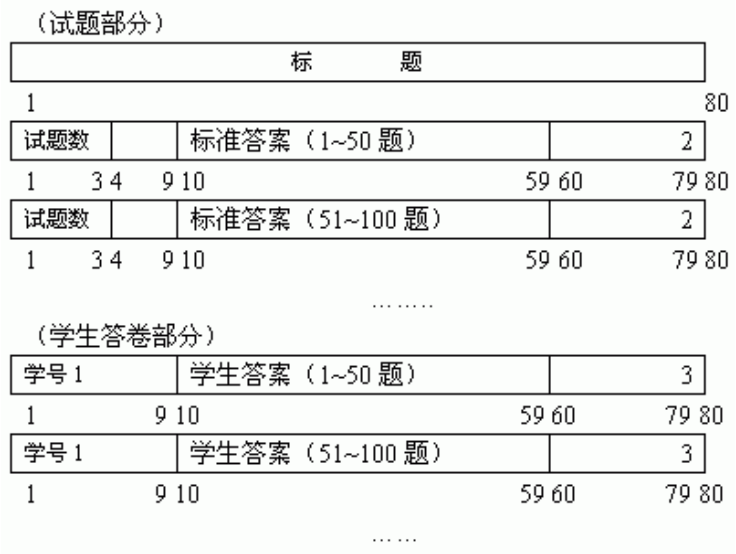
c1:a<b+c?	F	T	T	T	T	T	T	T	T	T	T
c2:b<a+c?	-	F	T	T	T	T	T	T	T	T	T

c3:c<a+b?	-	-	F	T	T	T	T	T	T	T	T
c4:$a^2+b^2=c^2$?	-	-	-	T	T	T	T	F	F	F	F
c5: $a^2+c^2=b^2$?	-	-	-	T	T	F	F	T	T	F	F
c6: $b^2+c^2=a^2$?	-	-	-	T	F	T	F	T	F	T	F
a1:非三角形	X	X	X								
a2:直角三角形							X		X	X	
a3:非直角三角形											X
a4:不可能				X	X	X		X			

根据该决策表设计测试用例如下：

用例 ID	*a*	*b*	*c*	预期输出
DT1	4	1	2	非三角形
DT2	1	4	2	非三角形
DT3	1	2	4	非三角形
DT4	?	?	?	不可能
DT5	?	?	?	不可能
DT6	?	?	?	不可能
DT7	3	4	5	直角三角形
DT8	?	?	?	不可能
DT9	3	5	4	直角三角形
DT10	5	3	4	直角三角形
DT11	2	3	4	非直角三角形

6．现有一个学生标准化考试批阅试卷，产生成绩报告的程序。其规格说明如下：程序的输入文件由一些有 80 个字符的记录组成，所有记录分为 3 组，如图：



（1）标题：该组只有一个记录，其内容是成绩报告的名字。

（2）各题的标准答案：每个记录均在第 80 个字符处标以数字 2。该组的记录：

第一个记录：第 1~3 个字符为试题数（1~999）。第 10~59 个字符是 1~50 题的标准答案（每个合法字符表示一个答案）。

第二个记录：是第 51~100 题的标准答案。

…….

（3）学生的答案：每个记录均在第 80 个字符处标以数字 3。每个学生的答卷在若干个记录中给出。

学号：1~9 个字符

1~50 题的答案：10~59。当大于 50 题时，在第二、三、……个记录中给出。

学生人数不超过 200，试题数不超过 999。

程序的输出有 4 个报告：

a)按学号排列的成绩单，列出每个学生的成绩、名次。

b)按学生成绩排序的成绩单。

c)平均分数及标准偏差的报告

d)试题分析报告。按试题号排序，列出各题学生答对的百分比。

采用边界值分析方法，分析和设计测试用例。分别考虑输入条件和输出条件，以及边界条件。采用错误推测法补充设计一些测试用例。

解：输入条件及相应的测试用例如下：

输入条件	测试用例
输入文件	空输入文件
标题	没有标题 标题只有一个字符 标题有 80 个字符
试题数	试题数为 1 试题数为 50 试题数为 51 试题数为 100 试题数为 0 试题数含有非数字字符
标准答案记录	没有标准答案记录，有标题 标准答案记录多于一个 标准答案记录少一个
学生人数	0 个学生 1 个学生 200 个学生 201 个学生
学生答题	某学生只有一个回答记录，但有两个标准答案记录 该学生是文件中的第一个学生 该学生是文件中的最后一个学生（记录数出错的学生）
学生答题	某学生有两个回答记录，但只有一个标准答案记录 该学生是文件中的第一个学生（记录数出错的学生） 该学生是文件中的最后一个学生
学生成绩	所有学生的成绩都相等 每个学生的成绩都不相等 部分学生的成绩相同 （检查是否能按成绩正确排名次） 有个学生 0 分 有个学生 100 分

输出条件及相应的测试用例表如下：

输出条件	测试用例
输出报告 a、b	有个学生的学号最小（检查按序号排序是否正确） 有个学生的学号最大（检查按序号排序是否正确） 适当的学生人数，使产生的报告刚好满一页（检查打印页数） 学生人数比刚才多出 1 人（检查打印换页）
输出报告 c	平均成绩 100 平均成绩 0 标准偏差为最大值（有一半的 0 分，其他 100 分） 标准偏差为 0（所有成绩相等）
输出报告 d	所有学生都答对了第一题 所有学生都答错了第一题 所有学生都答对了最后一题 所有学生都答错了最后一题 选择适当的试题数，是第四个报告刚好打满一页 试题数比刚才多 1，使报告打满一页后，刚好剩下一题未打

采用错误推测法还可补充设计一些测试用例：

- 程序是否把空格作为回答
- 在回答记录中混有标准答案记录
- 除了标题记录外，还有一些的记录最后一个字符即不是 2 也不是 3
- 有两个学生的学号相同
- 试题数是负数

第 4 章　白盒测试方法

1．简述白盒测试用例的设计方法，并进行分析总结。

解：白盒测试用例设计方法主要有逻辑覆盖和独立路径测试。

从覆盖源程序语句的详尽程度分析，逻辑覆盖主要有以下不同的覆盖标准：语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖、条件组合覆盖和路径覆盖。实际项目中，由于程序内部的逻辑存在不确定性和无穷性，尤其对于大规模复杂软件，不必采用所有的覆盖指标，而应根据实际情况选择合适的覆盖指标。

独立路径测试是在程序控制流图的基础上，通过分析控制结构的环路复杂性，导出可执行的独立路径集合，从而设计出相应的测试用例。设计出的测试用例要保证被测程序的每条可执行的独立路径至少被执行一次。独立路径测试给出了满足路径覆盖指标所需测试用例的下限，同时给出了语句覆盖的上限，它可以确保对所有相互独立的决策结果进行测试。

2．分析归纳逻辑覆盖的各种策略，并比较每种覆盖的特点，分析在怎样的情况下采用何种覆盖方式。

解：语句覆盖是选择足够多的测试数据，使被测程序中每个语句至少执行一次。语句覆盖是最弱的逻辑覆盖标准。

判定覆盖又叫分支覆盖，它不仅每个语句必须至少执行一次，而且每个判定表达式的每种可能的结果都应该至少执行一次。判定条件覆盖比语句覆盖强，但是对程序逻辑的覆盖程度仍然不高。

条件覆盖的含义是，使判定表达式中的每个条件都取到各种可能的结果。条件覆盖通常比判定覆盖强，但是也可能有相反的情况：虽然每个条件都取到了两个不同的结果，判定表达式却始终只取一个值。

判定/条件覆盖的含义是，选取足够多的测试数据，使得判定表达式中的每个条件都取到各种可能的值，而且每个判定表达式也都取到各种可能的结果。但有时判定/条件覆盖也并不比条件覆盖更强。

条件组合覆盖是更强的逻辑覆盖标准，它要求选取足够的测试数据，使得每个判定表达式中条件的各种可能组合都至少出现一次。满足条件组合覆盖标准的测试数据，也一定满足判定覆盖、条件覆盖和判定/条件覆盖标准。因此，条件组合覆盖是前述几种覆盖标准中最强的。但是，满足条件组合覆盖标准的测试数据并不一定能使程序中的每一条路径都执行到。

路径覆盖的定义是选取足够多测试数据，使程序的每一条可能路径都至少执行一次。但在实际问题中，一个不太复杂的程序，其路径数都可能是一个庞大的数字，以致要在测试中覆盖所有的路径是不可能实现的。即使对于路径数有限的程序做到了路径覆盖，也不能保证被测程序的正确性。

3．对图所示程序段进行语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖、条件组合覆盖和路径覆盖方法进行测试用例设计。

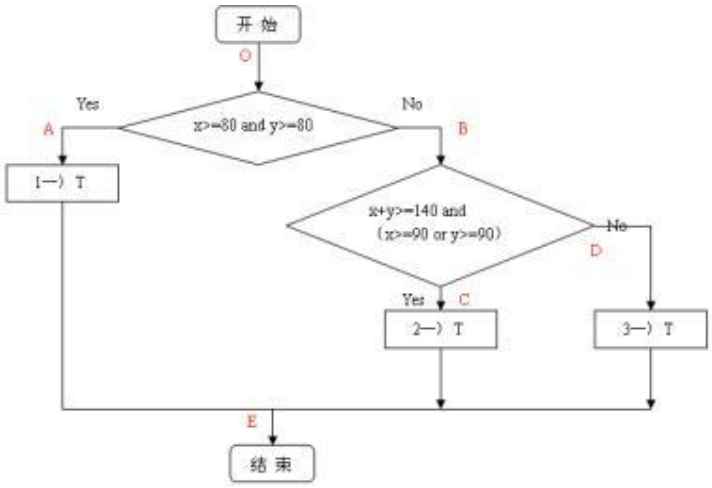


图 4.14 练习题 3

解：语句覆盖：x=90,y=90
x=79,y=90
x=70,y=60

判定覆盖：同上

条件覆盖：同上

判定/条件覆盖：同上

条件组合覆盖：**错误！未找到引用源。** x>=80,y>=80
错误！未找到引用源。 x>=80,y<80
错误！未找到引用源。 x<80,y>=80
错误！未找到引用源。 x<80,y<80
错误！未找到引用源。 x>=90,y>=90,x+y>=140
错误！未找到引用源。 x>=90,y<90,x+y>=140
错误！未找到引用源。 x<90,y>=90,x+y>=140
错误！未找到引用源。 x<90,y<90,x+y>=140
错误！未找到引用源。 x>=90,y>=90,x+y<140 不存在
错误！未找到引用源。 x>=90,y<90,x+y<140
错误！未找到引用源。 x<90,y>=90,x+y<140
错误！未找到引用源。 x<90,y<90,x+y<140
x=90,y=90
x=90,y=70
x=70,y=90
x=70,y=70
x=100,y=30
x=30,y=100
x=80,y=50

路径覆盖：同语句覆盖

4．请下述语句按照各种覆盖方法设计测试用例。

if (a>2 && b<3 && (c>4 || d<5))

```
 {
    statement;
}
else
{
    statement;
}
```

解：语句覆盖：a=3,b=2,c=5,d=5
a=2,b=2,c=5,d=5

判定覆盖：同上

条件覆盖：a=3,b=2,c=5,d=5
a=2,b=4,c=3,d=4

判定/条件覆盖：同条件覆盖

条件组合覆盖：错误！未找到引用源。 a>2,b<3,c>4,d<5
错误！未找到引用源。 a>2,b<3,c>4,d>=5
错误！未找到引用源。 a>2,b<3,c<=4,d<5
错误！未找到引用源。 a>2,b<3,c<=4,d>=5
错误！未找到引用源。 a>2,b>=3,c>4,d<5
错误！未找到引用源。 a>2,b>=3, c>4,d>=5
错误！未找到引用源。 a>2,b>=3, c<=4,d<5
错误！未找到引用源。 a>2,b>=3, c<=4,d>=5
错误！未找到引用源。 a<=2,b<3,c>4,d<5
错误！未找到引用源。 a<=2, b<3,c>4,d>=5
错误！未找到引用源。 a<=2, b<3,c<=4,d<5
错误！未找到引用源。 a<=2, b<3,c<=4,d>=5
错误！未找到引用源。 a<=2, b>=3,c>4,d<5
错误！未找到引用源。 a<=2, b>=3, c>4,d>=5
错误！未找到引用源。 a<=2, b>=3, c<=4,d<5
错误！未找到引用源。 a<=2, b>=3, c<=4,d>=5
测试数据略

5．针对 test 函数按照基本路径测试方法设计测试用例。

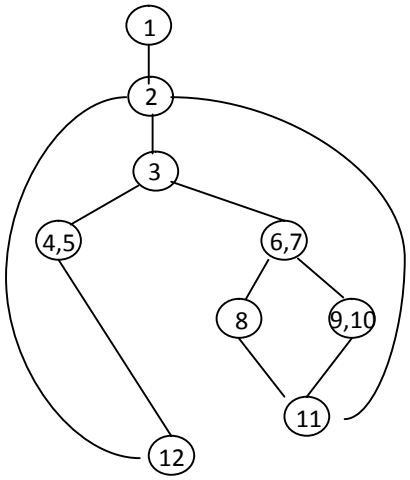
```
int Test(int i_count, int i_flag)
    {
      int i_temp = 0;
      while (i_count>0)
      {
        if (0 == i_flag)
        {
          i_temp = i_count + 100;
          break;
        }
        else
        {
```

```
            if (1 == i_flag)
            {
               i_temp = i_temp + 10;
            }
            else
            {
              i_temp = i_temp + 20;
            }
          }
          i_count--;
        }
        return i_temp;
      }
```

解：

```
int Test(int i_count, int i_flag)
    {
1       int i_temp=0;
2       while (i_count>0)
        {
3          If (0==i_flag)
           {
4             i_temp=i_count+100;
5             break;
           }
6          else
           {
7             If (1==i_flag)
              {
8                i_temp=i_temp+10;
              }
9             else
              {
10               i_temp=i_temp+20;
              }
           }
11         i_count--;
        }
12      return i_temp;
    }
```

程序控制流图：



程序环路复杂度：CC=4

基本路径集：path1　1-2-3-6-7-8-11-2-12

Path2　1-2-12

Path3　1-2-3-4-5-12

Path4　1-2-3-6-7-9-10-11-2-12

设计测试用例：

用例 ID	i_count	i_flag	预期输出
test1	1	1	10
test 2	0	2	0
test 3	2	0	102
test 4	1	3	20

6．对如图 4.15 所示的 N-S 图，计算所需的最少测试用例数。

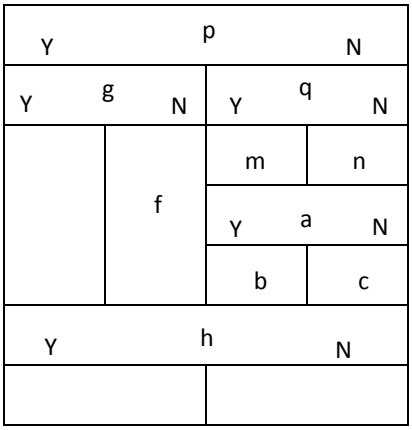


图 4.15 练习题 6

解：（2+2*2）*2=12

第 5 章　软件测试管理及自动化测试基础

1．简述软件测试自动化的意义。

解：软件测试自动化的意义是：（1）提高测试效率；（2）降低对软件新版本进行回归测试的开销；（3）完成手工测试不能或难以完成的测试；（4）具有一致性和可重复性；（5）更好地利用资源；（6）降低风险，增加软件信任度。

2．在运用软件自动化测试时，应注意哪些缺点和事项？

解：应注意：（1）软件自动化测试可能降低测试的效率；（2）测试首次运行时，可能发现大量错误，但当进行过多次测试后，发现错误的机率会相对较小，除非对软件进行了修改或在不同的环境下运行；（3）如果缺乏测试经验，测试的组织差、文档少或不一致，则自动化测试的效果比较差；（4）技术问题。作为第三方的技术产品，如果不具备解决问题的能力和技术支持或者产品适应环境变化的能力不强，将使得软件自动化工具的作用大大降低。

3．软件测试工具主要分为哪个大类？

解：根据测试方法不同，分为白盒测试工具和黑盒测试工具。

根据测试的对象和目的，分为单元测试工具、功能测试工具、负载测试工具、性能测试工具和测试管理工具等。

4．了解时下常用的自动化测试用具，并对这些工具进行针对性说明。

解：略。

5．简述软件测试管理过程。

解：首先由一位对整个系统设计熟悉的设计人员编写测试大纲，明确测试的内容和测试通过的准则，设计完整合理的测试用例，以便系统实现后进行全面测试。

然后在实现组将所开发的程序经验证后，提交测试组，由测试负责人组织测试，测试一般可按下列方式组织: （1）测试人员仔细阅读有关资料，包括规格说明、设计文档、使用说明书及在设计过程中形成的测试大纲、测试内容及测试的通过准则，全面熟悉系统，编写测试计划，设计测试用例，作好测试前的准备工作。（2）为了保证测试的质量，将测试过程分成几个阶段，即：代码审查、单元测试、集成测试、确认测试和系统测试。

6．简述软件测试管理的主要功能。

解：软件测试管理的主要功能是：测试控制对象的编辑和管理；测试流程控制和管理；统计分析和决策支持

7．试选择一个小型的应用系统，做功能测试计划并设计测试用例。

解：略。

第 6 章　WinRunner 测试工具

1．列举几种 WR 学习软件 GUI 的不同方式。

解：（1）使用 RapidTest Script wizard 学习软件每个窗体中所有 GUI 对象的属性。

（2）通过录制脚本的方法学习被录制的那部分软件中所有的 GUI 对象的属性。

（3）使用 GUI Map Editor 学习单个 GUI 对象、窗体或某个窗体中所有 GUI 对象的属性。

2．分别简述 WR 中同步点和检查点的作用。

解：当测试人员执行测试时，所测试的应用程序每次操作的响应时间并不一定，有时快，有时慢，导致执行输入动作的时间也需要等待。在测试脚本中插入同步点，当 WinRunner 执行到同步点时，会暂停执行以等待应用程序某些状态的改变后，再继续执行，以避免应用程序响应的时间超过 WinRunner 等待的时间而导致测试执行失败。

设定检查点可以检查所设定区域的显示是否和预期结果相符。通过检查点的设置以及对各点处输出信息的编程定义，可以在脚本运行结果单中查看各项测试内容是否都已通过。在功能测试中，检查点可以用在以下两个方面：检查应用程序经过修改后对象状态是否发生变化；检查对象数据是否和预期数据一致。

3．比较 WinRunner 中 GUIde Map File per Test 和 Global GUI Map File 两种模式的区别。

解：

两种模式	GUI Map File per Test	Global GUI Map File
方法	在测试的过程中将自动保 GUI 信息，打开测试时可以自动加载 GUI 文件	在测试的过程中需要保存 GUI，当应用程序改变时必须更新 GUI 文件
优点	1. 每个测试都有自己的 GUI 文件 2.不必保存或加载 GUI 3. 维护和修改简单（重录一次即可）	1. 当对象或窗体的描述改变，只需把 GUI 文件里对应的属性作相应的修改 2. 容易维护和更新（无须重录）
缺点	只要应用程序的 GUI 改变，每个测试的 GUI 文件都要重录或修改	当新建 GUI 或运行测试脚本时必须保存或装载 GUI 文件
建议	适用于初学者或被测软件的 GUI 不会产生变化	适用于经验丰富的 WinRunner 使用者，或被测软件的 GUI 可能会经常产生变化

4．简述利用 WinRunner 进行测试的过程可分为哪几个阶段，即操作步骤是什么？

解：WR 的测试过程分为以下六个阶段：（1）创建 GUI map；（2）创建测试；（3）调试测试；（4）执行测试；（5）查看测试结果；（6）报告发现的错误。

5．给出 WinRunner 中将测试脚本转换为数据驱动测试脚本的一种实现步骤。

解：可以通过下列步骤将测试脚本转换成数据驱动测试脚本：（1）加上开启及关闭数据表的指令；（2）加上循环并读取数据表的每一笔数据；（3）将录制的固定值与检查点的值参数化为数据表的字段值。

6．仿照实例 4，在 Flight Reservation 样本软件的 Flight 4B 版本中建立 GUI 对象检查点。

解：略。

7．仿照实例 5，在 Flight Reservation 样本软件的 Flight 4B 版本中建立图像检查点。

解：略。

8．仿照实例 8，在 Flight Reservation 样本软件的 Flight 4B 版本中练习文字检查点的应用。

解：略。

9．仿照实例 8，在 Flight Reservation 样本软件的 Flight 4B 版本中执行批次测试。

解：略。

10．仿照计算器加法功能的测试，完成对 Windows 的计算器减法、乘法和除法的测试。

解：略。

11．思考利用 WR 测试网易邮箱的登录模块。

解：略。

第 7 章　LoadRunner 测试工具

1. 试用 LoadRunner 所给的示例，根据自己的理解设计测试，制定负载测试计划、开发负载测试脚本、创建运行场景、运行测试以及依据结果利用 Analysis 分析结果。

解：略

2. 如何利用 LoadRunner 判断 HTTP 服务器的返回状态。

解：可以利用LR的内置函数web_get_int_property判断HTTP服务器的返回状态。例如：

```
#include "web_api.h"
  Action()
  {
    int HttpRetCode;
    web_url("my_home","URL=http://myhomeurl","TargetFrame=_TOP", LAST);
    HttpRetCode = web_get_int_property(HTTP_INFO_RETURN_CODE);
    if (HttpRetCode == 200)
     lr_log_message("The scrīpt successfully accessed the My_home home page");
    else
     lr_log_message("The scrīpt failed to access the My_home home page ");
     return 0;
  }
```

3. 一个公司的系统上线以后，用户分布在各个不同的地区，而且接入系统的方式和带宽也不同，这种情况下进行性能测试，如何保证更加真实的模拟用户行为？用 LoadRunner 可以做到吗？

解：可以。在Visual User Generator里面可以通过RTS（runTimeSetting）来模拟一个单个用户更加真实的行为，比如思考时间、网络带宽、是否清除cache等，同样的也可在场景中进行设置。而且LoadRunner提供设置不同用户组不同RunTimeSetting的功能。以达到模拟不同用户行为的更加真实组合。例如：假设有三种不同带宽的用户，而且上传和下载的带宽也有所不同，那么可以录制两个脚本，分别模拟上传和下载的用户行为，再在Controller里面，建立六个不同的脚本组，脚本组的用户数可以按照绝对或者百比分的方法分布。比如100，50，200用户或者20%，40%，40%等。然后设置不同的带宽和分布情况。这样不同用户组的虚拟用户模拟出来的就是不同带宽的用户实际接入情况。

4. 在 web 应用下，模拟十个用户并发进行数据的添加，结果每次执行全部成功，但是数据却不是十条，每次数据不一样，但是都比十小。这种情况产生的原因是什么？

解：是数据库的问题。大多数的数据库都有记录锁的问题，第一次的数据操作没有commit之前，第二次对同样表进行的操作可能就没有办法成功，所以每次数据的条数都达不到十条。

又因为每次的操作服务器的响应时间是不同的，所以不同虚拟用户的提交时间也是不同的，这样就导致每次提交成功的数据量不一致，导致每次结果的条数可能是不同的。

5. 在 LoadRunner 下如何让多个场景轮流执行？

解：为每个场景设置一个Group。点击Edit Schedule->选择Schedule by Group->设置Start when group XXX finishes，就可以实现多个场景轮流执行。

6. 请解释 LoadRunner 下最大并发用户数、业务操作响应时间、服务器资源监控指标的含义与用途。

解：最大并发用户数是指应用系统在当前环境下能承受的最大并发的用户数。用来考察某系统的最大负载；在 LoadRunner“事务性能摘要”图中可以获得业务操作的响应时间最大值、最小值和平均值，重点用于确定在方案执行期间响应时间过长的事务；

服务器资源监控指标包括内存和处理器。

内存：Linux 资源监控中指标内存页交换速率(Paging rate)，如果该值偶尔走高，表明当时有线程竞争内存。如果持续很高，则内存可能是瓶颈。也可能是内存访问命中率低。实际测试中，当并发点击数出现突然剧增前后，内存的 PR 值则居高 25 不下。说明目前测试的系统中内存存在瓶颈!

处理器：Linux 资源监控中指标 CPU 占用率持续超过 80%(对该值的要求，根据具体应用和机器配置而要求不同，有资料表明 95%)，表明瓶颈是 CPU。实际测试中，当并发点击数出现突然增加前后，CPU 的占用率持续保持在 86%以上!

第 8 章　JUnit

1．简述 JUnit 单元测试步骤。

解：1）判断组件的功能：通过定义应用的整体需求，然后将系统划分成几个对象；

2）设计组件行为：依据所处理的过程，可以通过一个正规或者非正规的过程实现组件行为的设计，可以使用UML或者其他文档视图来设计组件行为，从而为组件的测试打下基础；

3）编写单元测试程序（或测试用例）确认组件行为：这个阶段应假定组件的编码已经结束而组件工作正常，需要编写单元测试程序来确定其功能是否和预定义的功能相同，测试程序需要考虑所有正常和意外的输入，以及特定的方法能产生的溢出；

4）编写组件并执行测试：首先创建类及其所对应的方法标识，然后遍历每个测试实例，为其编写相应代码使其顺利通过，然后返回测试。继续这个过程直至所有实例通过；

5）测试替代品：对组件行为的其他方式进行考虑，设计更周全的输入或者其他错误条件，编写测试用例来捕获这些条件，然后修改代码使得测试通过；

6）重整代码：如果有必要，在编码结束时对代码进行重整和优化，改动后返回单元测试并确认测试通过；

7）当组件有新的行为时，编写新的测试用例：每次在组件中发现故障，编写一个测试实例重复这个故障，然后修改组件以保证测试实例通过。同样，当发现新的需求或已有的需求改变时，编写或修改测试实例以响应此改变，然后修改代码；

8）代码修改，重复测试：每次代码修改时，重复所有的测试以确保没有打乱代码。

2．对下列代码进行单元测试。

Triangle.java

```
public class Triangle
{
    // 定义三角形的三边
    protected long lborderA = 0;
    protected long lborderB = 0;
    protected long lborderC = 0;
    // 构造函数
    public Triangle(long lborderA, long lborderB, long lborderC)
    {
        this.lborderA = lborderA;
        this.lborderB = lborderB;
        this.lborderC = lborderC;
    }
    /*
     * 判断是否是三角形
```

```
     */
    public boolean isTriangle(Triangle triangle) {
        boolean isTriangle = false;
        // 判断边界，大于 0  小于 200，出界返回 false
        if ((triangle.lborderA > 0 && triangle.lborderA <200)
           && (triangle.lborderB > 0 && triangle.lborderB <200)
           && (triangle.lborderC > 0 && triangle.lborderC <200))
         {
            // 判断两边之和大于第三边
            if ((triangle.lborderA < (triangle.lborderB + triangle.lborderC))
               && (triangle.lborderB < (triangle.lborderA + triangle.lborderC))
               && (triangle.lborderC < (triangle.lborderA + triangle.lborderB))) {
             isTriangle = true;
            }
    Return isTrue;
        }
    /**
     * 判断三角形类型
     *
    public String getType(Triangle triangle) {
        String strType = "";
        // 判断是否是三角形
        if (this.isTriangle(triangle))
        {
            // 判断是否是等边三角形
            if (triangle.lborderA == triangle.lborderB
               && triangle.lborderB == triangle.lborderC)
                strType = "等边三角形";
            // 判断是否是不等边三角形
            else if ((triangle.lborderA != triangle.lborderB) &&
                    (triangle.lborderB != triangle.lborderC) &&
                    (triangle.lborderA != triangle.lborderC))
                strType = "不等边三角形";
              else
                strType = "等腰三角形";
        }
        return strType;
    }
}
```

解：具体步骤为：

1）新建 sample 项目，源文件夹为 src；

2）新建并实现Triangle 类（代码如题），包名为com.ime.sample；

3）新建源文件夹 test；

4）新建并实现TriangleTest 类（代码如下），包名为com.ime.sample.test；

5）使用JUnit 运行TriangleTest。

TriangleTest 类：

```
import junit.framework.Test;
import junit.framework.TestCase;
import junit.framework.TestSuite;

/**
 * Tests about <code>Triangle</code> class.
 *
 * @author yangyz@NEUSOFT
 */
public class TriangleTest extends TestCase {
	public Triangle triangle = null;
	public static Test suite() {
		return new TestSuite(TriangleTest.class);
	}

	/**
	 * Test for Triangle#isTriangle().<p>
	 *
	 * Setup: lborderA = 0; lborderB = 3; lborderC = 3<br>
	 * Expected: method returns false
	 */
	public void testIsTriangle01() {
		// setup
		triangle = new Triangle(0, 3, 3);
		// expected
		boolean expected = false;
		// execute
		boolean result = Triangle.isTriangle(triangle);
		// actual
		boolean actual = result;
		// assert
		assertEquals("While a=0; b=3; c=3 mehtod fails!", expected, actual);
	}

	/**
	 * Test for Triangle#isTriangle().<p>
	 *
	 * Setup: lborderA = 3; lborderB = 3; lborderC = 3<br>
	 * Expected: method returns true
	 */
```

```
public void testIsTriangle02() {
        // setup
        triangle = new Triangle(3, 3, 3);
        // expected
        boolean expected = true;
        // execute
        boolean result = Triangle.isTriangle(triangle);
        // actual
        boolean actual = result;
        // assert
        assertEquals("While a=3; b=3; c=3 mehtod fails!", expected, actual);
}

/**
 * Test for Triangle#getType().<p>
 *
 * Setup: lborderA = 0; lborderB = 3; lborderC = 3<br>
 * Expected: method returns "不是三角形"
 */
public void testGetType01() {
        // setup
        triangle = new Triangle(0, 1, 1);
        // expected
        String expected = "不是三角形";
        // execute
        String result = Triangle.getType(triangle);
        // actual
        String actual = result;
        // assert
        assertEquals("While a=0; b=3; c=3 mehtod fails!", expected, actual);
}

/**
 * Test for Triangle#getType().<p>
 *
 * Setup: lborderA = 3; lborderB = 4; lborderC = 5<br>
 * Expected: method returns "不等边三角形"
 */
public void testGetType02() {
        // setup
        triangle = new Triangle(3, 4, 5);
        // expected
        String expected = "不等边三角形";
        // execute
```

```
        String result = Triangle.getType(triangle);
        // actual
        String actual = result;
        // assert
        assertEquals("While a=3; b=4; c=5 mehtod fails!", expected, actual);
}

/**
 * Test for Triangle#getType().<p>
 *
 * Setup: lborderA = 3; lborderB = 3; lborderC = 3<br>
 * Expected: method returns "等边三角形"
 */
public void testGetType03() {
        // setup
        triangle = new Triangle(3, 3, 3);
        // expected
        String expected = "等边三角形";
        // execute
        String result = Triangle.getType(triangle);
        // actual
        String actual = result;
        // assert
        assertEquals("While a=3; b=3; c=3 mehtod fails!", expected, actual);
}

/**
 * Test for Triangle#getType().<p>
 *
 * Setup: lborderA = 3; lborderB = 3; lborderC = 5<br>
 * Expected: method returns "等腰三角形"
 */
public void testGetType04() {
        // setup
        triangle = new Triangle(3, 3, 5);
        // expected
        String expected = "等腰三角形";
        // execute
        String result = Triangle.getType(triangle);
        // actual
        String actual = result;
        // assert
        assertEquals("While a=3; b=3; c=5 mehtod fails!", expected, actual);
}
```

```
    // 执行测试
    public static void main(String[] args) {
        // Text ui 方式
        // junit.textui.TestRunner.run(suite());
        // Swing ui 方式
        junit.swingui.TestRunner.run(TriangleTest.class);
        // AWT ui 方式
        // junit.awtui.TestRunner.run(TriangleTest.class);
    }
}
```