软件学院 06、07、08 级软件工程专业(2009-2)

《软件测试》期末试题试卷(B)

(考试形式：开　卷　　考试时间：2 小时)

警示

《中山大学授予学士学位工作细则》第六条

考试作弊不授予学士学位

方向：_____　姓名：_____　学号：_____成绩：_____

一、三角形测试(30 分)

假设你被告知去测试一个很简单的程序。这个程序接受 3 个整数作为输入或者参数。这些整数表示一个三角形的各个边长。该程序会打印“等边”、“等腰”和“不规则”。

你的任务是写一个有效且高效的测试用例集进行功能性和用户界面测试。对于有效而言，指的是这个测试集应该找到常见的缺陷。而对于高效而言，指的是这个测试集应该在合理的工作量内找到这些缺陷。对于测试用例而言，指的是测试者的动作、数据和期望结果。这个动作通常会输入一些数据，但也可能是其他的一些动作。这里提供了一个解答模板。我建议你使用这个解答格式。

表 1 三角形测试解答模板

测试者行为和数据	期待结果

二、使用 Wiegers 列表的三角形需求复审（20 分）

假设你已经收到了如下关于三角形分类程序的需求规约。

该程序可以从一个用户界面友好的方式接受三个整型数输入，表示三角形的边长。它应该很快画出这个三角形，并输出“不规则”、“等腰”、或者“等边”。

下面是 wiegers 的 10 大需求问题列表。

1.省略了关键需求，包括业务、用户、质量、功能和非功能需求。

2.不充足的用户介入。当然，我们并不总是能和客户交谈，所以你可能需要依靠用户代言人诸如销售、市场和技术支持人员。

3.含混和不明确性。在我的经验中这是一个普遍存在的问题。

4.没有为需求划分优先级。如果需求冲突或者我们想按照优先级序列来建造和测试特性时，这就特别成问题。

5.提供了没有人使用或者需要的功能。这是一个许多工程师会发现无法抵御的强烈念头。好的设计提供了未来的扩充性，但是好的项目经理只确保我们刚好提供了当前所需要的那些功能。

6.分析瘫痪。你需要多少信息来最终决定是否包括一个特性？现在不马上做这个决定会有什么样的风险以及与之相关的代价？

7.规模蔓延。你曾经在这样一个项目中工作过吗？只是由于听到某人说“我们只是再加上一个特性、能力、报告、屏幕，等等”，于是原本可以冻结的需求就无法被冻结。记住，一个延误了的项目完成日期，可能由于几十个小的交付日期被延误而造成，也可能只是由于延误了一个重要项目交付日期。

8.不充足的需求变化控制过程。这也可以适用于在需求冻结之后，人们颠覆了变化控制过程，偏移到自己的工作上。

9.不充分的变化影响分析。如果我们决定对需求做一个后期变动，在编程工作上有什么影响？在测试上呢？对配置管理、技术出版、项目管理和其他支持小组上的影响呢？当考虑过所有的风险和代价之后，变动获得的机遇和利益真的能超过风险和代价吗？

10.不充足的版本控制。如果你听到某人问类似“产品中有什么？”的问题，你可以确定需求和设计规约的版本控制中有问题。

你通过使用 Wiegers 的十大需求问题，从这个需求规约中来识别尽可能多的问题。

表 **2** 问题描述

有问题的措辞	**Wiegers** 问题编号

表 3 问题解决

改善的措辞	解决的问题编号

三、ATM 状态转换测试（30 分）

假设你在测试一个只支持取款的 ATM 机，该 ATM 机接受其所在网络中任何一家银行的卡。ATM 会验证客户的 PIN，然后允许取款。它分发现钞并在交易结束时立刻退卡给客户。客户可以取消而不是输入 PIN，或撤销取款要求。如果这样，ATM 机会退卡给用户。ATM 机也可以在三次无效的 PIN 尝试后退卡给用户。

对于这个练习，为 ATM 机画出状态图，而后将状态图翻译为测试用况。

四、关于十六进制字符输入转换的控制流测试（20 分）

现在把白盒控制流测试概念运用到一个简单程序——把十六进制字符输入转换到一个整型数的简单的 C 程序，如程序清单 1 所示，它显示了所转换的十六进制字符的数量以及它们的值（以 16 进制显示）。它忽略了非十六进制字符。如果你有一个可用的编辑器，该程序可以运行于 Windows 或者 Linux 系统上。

本练习有两个部分。

1. 理解潜在的测试用况的数量
2. 为语句、分支、条件和循环的全覆盖创建一套测试用例集

程序清单 1 十六进制数转换程序

```
1.main()
2./*Convert hex digits to a number*/
3.{
4.    int c;
5.    unsigned long int hexnum, nhex;
6.
7.    hexnum=nhex=0;
8.
9.    while((c=getchar())!=EOF){
10.   switch(c){
11.   case'0':case'1':case'2':case'3':case'4':
12.   case'5':case'6':case'7':case'8':case'9':
13.         /*convert a decimal digit*/
14.         nhex++;
15.         hexnum *=0x10;
16.         hexnum +=(c-'0');
17.         break;
18.         case'a':case'b':case'c':
```

```
19.         case'd':case'e':case'f':
20.         /*convert a lower case hex digit*/
21.         nhex++;
22.         hexnum*=0x10;
23.         hexnum+=(c-'a'+0xa);
24.         break;
25.         case'A':case'B':case'C':
26          case'D':case'E':case'F':
27.         /*convert an upper case hex digit*/
28.         nhex++;
29.         hexnum*=0x10;
30.         hexnum+=(c-'A'+0xA);
31.         break;
32.         default:
33.         /*Skip any non-hex characters*/
34.         break;
35.         }
36.   }
37    printf("Got %d hex digits: %x\n", nhex, hexnum);
38.   return 0;
39.}
```