软件质量保证与测试

Software Quality Assurance and Testing

第 5 章 软件测试过程

5.1 单元测试

5.1.2 单元测试的任务

金陵科技學院

单元测试的步骤

代码编写完成后的单元测试工作，主要分为两个步骤：静态检查和动态测试。

静态检查是单元测试的第一步，这个阶段的工作主要是保证代码算法的逻辑正确性，应通过人工检查发现代码的逻辑错误，其次还要检查代码的清晰性、规范性、一致性，反复执行的代码，还需要分析算法是否高效。

第二步是通过设计测试用例，执行被测程序，比较实际结果与预期结果的异同，以发现程序中的错误。

单元测试的步骤

经验表明，使用静态检查法能够有效的发现30%到70%的逻辑设计和编码错误。但是代码中仍会有大量的隐性错误无法通过静态检查发现，必须通过动态测试才能够捕捉和发现。

动态测试是单元测试的重点与难点。

单元测试的任务

一般而言，应当对程序模块进行以下动态单元测试：

1、对模块内所有独立的执行路径至少测试一次；

2、对所有的逻辑判定，取“真”与“假”的两种情况都至少执行一次；

3、在循环的边界和运行界限内执行循环体；

4、测试内部数据的有效性等。

单元测试的任务

单元测试的依据是软件的详细设计和编码标准等，检查和测试对象主要就是源程序。

单元测试的任务主要包括：

① 验证代码能否达到详细设计的预期要求。

② 发现代码中不符合编码规范的地方。

③ 准确定位错误，以便排除错误。

具体而言，单元测试应检查和测试的内容如下。

1. 算法和逻辑

检查算法和内部各个处理逻辑的正确性，例如：某程序员编写打印下降三角形九九乘法表的程序如下：

```
*** 九九打印乘法表
    for i=1 to 9
        for j=1 to 9
          print i, ”*”, j , “=”, i*j
        endfor
        print enter
    endfor
```

通过检查和测试应能发现程序逻辑是错误的，打印出来的不是下降三角形的九九乘法表，而是9*9的方阵。

1. 算法和逻辑

通过检查和测试应能发现程序逻辑是错误的，打印出来的不是下降三角形的九九乘法表，而是9*9的方阵。

改正的办法是把第二个循环的

```
for j=1 to 9
```

改为:

```
for j=1 to i
```

2. 模块接口

对模块自身的接口做正确性检查；确定形式参数个数、数据类型、顺序是否正确；确定返回值类型，检查返回值的正确性。

检查调用其他模块的代码的正确性；调用其他模块时给定的参数类型正确与否、参数个数正确与否、参数顺序正确与否，特别是具有多态的方法尤其需要注意。检查返回值正确与否，有没有误解返回值所表示的意思。必要时可以对每个被调用的方法的返回值用显式代码如程序插桩，作正确性检查，如果被调用方法出现异常或错误程序应该给予反馈，并添加适当的出错处理代码。

2. 模块接口

例如：某程序员编写的求平均成绩的代码段如下：

```
Function score_average (char a[i])
        S=0
        Av=0
        for i=1 to a.lenth
            s=s+a[i]
        endfor
        Av=s / a.lenth
    endfunction
```

2. 模块接口

程序中的问题是：

函数内部把成绩当成数值型数据来处理，直接进行累加，而形式参数中存放成绩的是字符型数组，所以接口和内部实现是不一致的。

要改正的话，既可以修改程序内部实现，也可以修改接口，但如果事先还没有把程序接口规定死的话，显然修改接口要比修改内部实现简单一些。

3.数据结构

检查全局和局部数据结构的定义（如：队列、堆栈等）是否能实现模块或方法所要求的功能。

例如某程序中需要实现先来先服务的任务调度，但为此定义的数据结构为栈，这显然是错误的，因为栈用于实现后进者先出。

改正的办法是定义一个队列，而不是栈。

4. 边界条件

检查各种边界条件发生时程序执行是否仍然正确，包括检查判断条件的边界等，例如，某程序用于实现将百分制成绩转换为五级计分制成绩，代码如下：

```
if cj>90 print “优秀”;
if cj<90 and cj>80 print “良好”;
if cj<80 and cj>70 print “中等”;
if cj<70 and cj>60 print “及格”;
if cj<60 print “不及格”;
```

4. 边界条件

显然，程序中的判断条件漏掉了相等的情况。

例如当 cj=90 时，程序不会给出转换结果。

改正的办法是在适当的位置，加上“=”。

5. 独立的路径

程序编写时可能存在疏漏，应对照程序详细设计书的要求对程序进行检查和测试，看是否漏掉了某些原本需要的处理逻辑，也就是少了某些应当有的独立路径，或者某些独立路径存在处理错误。例如，某程序用于实现将百分制成绩转换为五级计分制成绩，代码如下：

```
if cj>=90 print “优秀”;
if cj<90 and cj>=80 print “良好”;
if cj<70 and cj>=60 print “及格”;
if cj<60 print “不及格”;
```

5. 独立的路径

对照程序详细设计书可以发现，程序漏掉了

cj<80 and cj>=70

这种情况。

当 cj<80 and cj>=70 时，程序无法给出转换结果。

6. 错误处理

单元模块应能预见某些代码运行时可能出错的条件和情况，并设置适当的出错处理代码，以便在相关代码行运行出现错误时，能妥善处理，保证整个单元模块处理逻辑的正确性，这种出错处理应当是模块功能的一部分。

6. 错误处理

例如，有代码段如下：

```
Procedure score_average ( )
        read a[i]
        S=0
        Av=0
        for i=1 to a.lenth
            s=s+a[i]
        endfor
        Av=s / a.lenth
    endfunction
```

6. 错误处理

这代码并不复杂，看上去似乎也没有什么问题，但实际上程序中的

read a[i]

这一代码行执行时存在出错的可能，例如 a[i] 可能并不存在。

6. 错误处理

为此，应设置适当的出错处理代码，以便在相关代码行运行出现错误时，能妥善处理，修改后的代码段如下：

```
Procedure score_average ( )
on error print “there is no a[i]”
    read a[i]
on error
S=0
Av=0
for i=1 to a.lenth
   s=s+a[i]
endfor
Av=s / a.lenth
endfunction
```

6. 错误处理

若出现下列情况之一，则表明模块的错误处理功能包含有错误或缺陷：

- 出错的描述难以理解；
- 出错的描述不足以对错误定位，不足以确定出错的原因；
- 显示的错误信息与实际的错误原因不符；
- 对错误条件的处理不正确；
- 在对错误进行处理之前，错误条件已经引起系统的干预等。

7. 输入数据

应当对输入数据进行正确性、规范性或者合理性检查，经验表明，没有对输入数据进行必要和有效的检查，是造成软件系统不稳定或者执行出问题的主要原因之一。

例如某成绩管理软件，在成绩输入模块，没有对输入的成绩数据进行合理性检查，某个同学的某门课程成绩应当是90分，一不小心输入成了900分，数据保存后，在后来的求平均成绩时，该同学的平均成绩高达200多分，这显然不符合情理，但出现这一错误后，首先想到的可能是数据统计出错了，这种问题的原因排查也很麻烦。

7. 输入数据

另外，在系统登录模块，如果不对用户名和密码输入的规范性和合理性进行检查，则恶意用户有可能采用注入式攻击等方式来试图非法进入系统。

8. 表达式、SQL语句

应检查程序中的表达式及SQL语句的语法和逻辑的正确性。对表达式应该保证不含二义性，对于容易产生歧义的表达式或运算符优先级，如： &&、||、++、-- 等，可以采用扩号“（ ）”运算符避免二义性，这样一方面能够保证代码执行的正确性，同时也能够提高代码的可读性。

8. 表达式、SQL语句

例如，&& 和 || 是有优先级差别的，&& 的优先级高于 ||，但这一点有时很容易被忽视，为此可以采用加“( )”的方式，来写相关的表达式。

例如：职称为工程师或讲师，并且年龄小于35岁，写成表达式应当是：

(ZC=“工程师”||ZC=“讲师”)&& NL<35

如果不加括号，那么表达式的意思就和要求不一致了。

9. 常量或全局变量的使用

应检查常量和全局变量的使用是否正确。明确所使用的常量或全局变量的数据类型；保证常量数据类型和取值的恒定性，不能前后不一致。另外还要特别注意有没有和全局变量同名的局部变量存在，如果有，要清楚它们各自的作用范围，不能混淆。

10. 标识符定义

标识符定义应规范一致；保证变量命名既简洁又能够见名知意，不宜过长或过短，各种标识符应规范、容易记忆和理解。

11. 程序风格

检查程序风格的一致性、规范性；代码必须符合企业规范，保证所有成员的代码风格一致、格式工整。

例如：对数组做循环时，不要一会儿采用下标变量从下到上的方式（如:for(I=0;I++；I<10)），一会儿又采用从上到下的方式（如:for(I=9;I--;I>=0)）；应该尽量采用统一的方式，要么统一从下到上，要么统一从上到下。

建议采用for循环和While循环，不要采用 do { } while循环等。

12. 注释

应检查注释是否完整；是否清晰简洁；是否正确的反映了代码的功能。错误的注释比没有注释更糟，会让程序阅读者产生错误的理解。应检查是否做了多余的注释；对于简单的一看就懂的代码没有必要注释。还有就是应检查对包、类、属性、方法功能、参数、返回值的注释是否正确且容易理解等。

本节内容就讲到这里，谢谢，再见！

金陵科技學院