7.7黑盒测试

黑盒测试着重测试软件功能。黑盒测试并不能取代白盒测试，它是与白盒测试互补的测试方法，它很可能发现白盒测试不易发现的其他类型的错误。

在黑盒测试力图中会发现下述类型的错误：

(1) 功能不正确或遗漏了功能；

(2) 界面错误；

(3) 数据结构错误或外部数据库访问错误；

(4) 性能错误；

(5) 初始化和终止错误。

问题一：黑盒测试和白盒测试分别用于测试的什么阶段

答案：白盒测试在测试过程的早期阶段进行，而黑盒测试主要用于测试过程的后期。

问题：我们在设计黑盒测试方案时，我们应该考虑什么问题？

答案：

（1）怎样测试功能的有效性？

（2）哪些类型的输入可构成好测试用例？

（3）系统是否对特定的输入值特别敏感？

（4）怎样划定数据类的边界？

（5）系统能够承受什么样的数据率和数据量？

（6）数据的特定组合将对系统运行产生什么影响？

7.1.1等价划分

等价划分的概念是把程序的输入域划分成若干个数据类，据此导出测试用例。等价划分法力图设计出能发现若干类程序错误的测试用例，从而减少必须设计的测试用例的数目。

黑盒测试选取少量最有代表性的输入数据作为测试数据，以期用较小的代价泰露出较多的程序错误。等价划分法力图设计出能发现若干类程序错误的测试用例,从而减少必须设计的测试用例的数目。等价类可以合理地减少测试，如果把所有可能的输入数据(有效的和无效的)划分成若干个等价类，则可以合理地做出下述假定：每类中的一个典型值在测试中的作用与这一类中所有其他值的作用相同。因此，可以从每个等价类中只取一组数据作为测试数据。这样选取的测试数据最有代表性，最可能发现程序中的错误。

划分等价类有一定的规则

(1) 如果规定了输入值的范围，则可划分出一个有效的等价类(输入值在此范围内)，两个无效的等价类(输入值小于最小值或大于最大值)。

(2) 如果规定了输入数据的个数，则类似地也可以划分出一个有效的等价类和两个无效的等价类。

(3) 如果规定了输入数据的一组值，而且程序对不同输入值做不同处理，则每个允许的输入值是一个有效的等价类，此外还有一个无效的等价类(任一个不允许的输入值)

(4) 如果规定了输入数据必须遵循的规则，则可以划分出一个有效等价类(符合规则)和若干个无效等价类(从各种不同角度违反规则)。

(5) 如果规定了输入数据为整型，则可以划分出正整数、零和负整数3个有效类。

(6) 如果程序的处理对象是表格，则应该使用空表，以及含一项或多项的表。

问题：在等价类进行分类以后，设计测试方案有两个步骤分别是什么？

答案： (1) 设计一个新的测试方案以尽可能多地覆盖尚未被覆盖的有效等价类，重复这一步骤直到所有有效等价类都被覆盖为止。

(2) 设计一个新的测试方案，使它覆盖一个而且只覆盖一个尚未被覆盖的无效等价类，重复这一步骤直到所有无效等价类都被覆盖为止。

有一点要注意的是通常程序发现一类错误后就不再检查是否还有其他错误，因此，应该使每个测试方案只覆盖一个无效的等价类。

接下来是一个例子：

问题：

假设有一个把数字串转变成整数的函数。运行程序的计算机字长16位，用二进制补码表示整数。这个函数是用Pascal语言编写的，它的说明如下：

function strtoint (dstr:shortstr):integer;

函数的参数类型是shortstr,它的说明是:

type shortstr=array［1..6］ of char;

被处理的数字串是右对齐的，也就是说，如果数字串比6个字符短，则在它的左边补空格。如果数字串是负的，则负号和最高位数字紧相邻(负号在最高位数字左边一位)。

考虑到Pascal编译程序固有的检错功能，测试时不需要使用长度不等于6的数组做实在参数，更不需要使用任何非字符数组类型的实在参数。

等价类划分：

分析这个程序的规格说明，可以划分出如下等价类。

有效输入的等价类有

(1) 1～6个数字字符组成的数字串(最高位数字不是零)。

(2) 最高位数字是零的数字串。

(3) 最高位数字左邻是负号的数字串。

无效输入的等价类有

(1) 空字符串(全是空格)。

(2) 左部填充的字符既不是零也不是空格。

(3) 最高位数字右面由数字和空格混合组成。

(4) 最高位数字右面由数字和其他字符混合组成。

(5) 负号与最高位数字之间有空格。

合法输出的等价类有

(1) 在计算机能表示的最小负整数和零之间的负整数。

(2) 零。

(3) 在零和计算机能表示的最大正整数之间的正整数。

非法输出的等价类有

(1) 比计算机能表示的最小负整数还小的负整数。

(2) 比计算机能表示的最大正整数还大的正整数。

因为所用的计算机字长16位，用二进制补码表示整数，所以能表示的最小负整数是-32 768，能表示的最大正整数是32 767

测试案例：

|  |  |  |  |
| --- | --- | --- | --- |
| **编号** | **描述** | **输入** | **预期输出** |
| 1 | 1～6个数字组成的数字串，输出是合法的正整数 | ‘1’ | 1 |
| 2 | 最高位数字是零的数字串，输出是合法的正整数 | ‘000001’ | 1 |
| 3 | 负号与最高位数字紧相邻，输出合法的负整数 | ‘-00001’ | -1 |
| 4 | 最高位数字是零，输出也是零 | ‘000000’ | 0 |
| 5 | 太小的负整数 | ‘-47561’ | 错误——无效输入 |
| 6 | 太大的正整数 | ‘132767’ | 错误——无效输入 |
| 7 | 空字符串 | ‘ ’ | 错误——没有数字 |
| 8 | 字符串左部字符既不是零也不是空格 | ‘×××××1’ | 错误——填充错 |
| 9 | 最高位数字后面有空格 | ‘12’ | 错误——无效输入 |
| 10 | 最高位数字后面有其他字符 | ‘1××2’ | 错误——无效输入 |
| 11 | 负号和最高位数字之间有空格 | ‘-12’ | 错误——负号位置错 |

7.7.2边界值分析

经验表明，处理边界情况时程序最容易发生错误。例如，许多程序错误出现在下标、纯量、数据结构和循环等等的边界附近。因此，设计使程序运行在边界情况附近的测试方案，暴露出程序错误的可能性更大一些。那么就提出了边界分析法，使用边界值分析方法设计测试方案首先应该确定边界情况，通常输入等价类和输出等价类的边界。选取的测试数据应该刚好等于、刚刚小于和刚刚大于边界值。

通常在设计测试方案时总是联合使用哪两种技术？

答案：等价划分和边界值分析

为了测试前述的把数字串转变成整数的程序，除了上一小节已经用等价划分法设计出的测试方案外，还应该用边界值分析法再补充下述测试方案。

|  |  |  |  |
| --- | --- | --- | --- |
| **编号** | **描述** | **输入** | **预期输出** |
| 1 | 使输出刚好等于最小的负整数 | ‘-32768’ | -32768 |
| 2 | 使输出刚好等于最大的正整数 | ‘32767’ | 32767 |
| 3 | 使输出刚刚小于最小的负整数 | ‘-32769’ | 错误——无效输入 |
| 4 | 使输出刚刚大于最大的正整数 | ‘32768’ | 错误——无效输入 |

7.7.3.错误推测

使用边界值分析和等价划分技术,有助于设计出具有代表性的、因而也就容易暴露程序错误的测试方案。但是,不同类型不同特点的程序通常又有一些特殊的容易出错的情况。此外,有时分别使用每组测试数据时程序都能正常工作,这些输人数据的组合却可能检测出程序的错误。一般说来,即使是一个比较小的程序,可能的输入组合数也往往十分巨大，因此必须依靠测试人员的经验和直觉，从各种可能的测试案中选出些此最好的。

错误推测法在很大程度上靠直觉和经验进行。它的基本想法是列举出程序中可能有的错误和容易发生错误的特殊情况，并且根据它们选择测试方案。

我们应该仔细分析程序规格说明书，注意找出其中遗漏或省略的部分，以便设计相应的测试方案，检测程序员对这些部分的处理是否正确。

在经验表明，在一段程序中已经发现的错误数目往往和尚未发现的错误数成正比。例如，在IBM OS/370操作系统中，用户发现的全部错误的47%只与该系统4%的模块有关。因此，在进一步测试时要着重测试那些已发现了较多错误的程序段。

等价划分法和边界值分析法都只孤立地考虑各个输入数据的测试功效，而没有考虑多个输入数据的组合效应，可能会遗漏了输入数据易于出错的组合情况。

问题：在选择输入组合的一个有效途径是利用---或----为工具，列出输入数据各种组合与程序应作的动作(及相应的输出结果)之间的对应关系，然后为判定表的每一列至少设计一个测试用例。其选择输入组合的另一个有效途径是把计算机测试和人工检查代码结合起来。

答案：判定表或判定树

7.8

调试（也称为纠错）作为成功测试的后果出现，即调试是在测试发现错误之后排除错误的过程。

7.8.1.调试过程

调试不是测试。

调试过程从执行一个测试用例开始，评估测试结果，如果发现实际结果与预期结果不一致，则这种不一致就是一个症状，它表明在软件中存在着隐藏的问题。调试过程试图找出产生症状的原因，以便改正错误。

7.8.2.调试途径

1.蛮干法

蛮干法按照“让计算机自己寻找错误”的策略，这种方法印出内存的内容，激活对运行过程的跟踪，并在程序中到处都写上WRITE（输出）语句，希望在这样生成的信息海洋的某个地方发现错误原因的线索。

2.回溯法

回溯是一种相当常用的调试方法，当调试小程序时这种方法是有效的。具体做法：从发现症状的地方开始，人工沿程序的控制流往回追踪分析源程序代码，直到找出错误原因为止。

3原因排错法

对分查找法、归纳法和演绎法都属于原因排除法。

对分查找法的基本思路是，如果已经知道每个变量在程序内若干个关键点的正确值，则可以用赋值语句或输入语句在程序中点附近“注入”这些变量的正确值，然后运行程序并检查所得到的输出。

归纳法是从个别现象推断出一般性结论的思维方法。使用这种方法调试程序时，首先把和错误有关的数据组织起来进行分析，以便发现可能的错误原因。然后导出对错误原因的一个或多个假设，并利用已有的数据来证明或排除这些假设。

演绎法从一般原理或前提出发，经过排除和精化的过程推导出结论。采用这种方法调试程序时，首先设想出所有可能的出错原因，然后试图用测试来排除每一个假设的原因

7.9 软件可靠性

7.9.1.基本概念

软件可靠性是程序在给定的时间间隔内，按照规格说明书的规定成功地运行的概率。软件可靠性随着给定的时间间隔的加大而减少。

软件可用性是程序在给定的时间点，按照规格说明书的规定，成功地运行的概率。

在可靠性和可用性之间的主要差别是，可靠性意味着在0到t这段时间间隔内系统没有失效，而可用性只意味着在时刻t，系统是正常运行的

7.9.2.估算平均无故障时间的方法

1.符号

在估算MTTF的过程中使用下述符号表示有关的数量。

ET——测试之前程序中错误总数；

IT——程序长度(机器指令总数)；

τ——测试(包括调试)时间；

Ed(τ)——在0至τ期间发现的错误数；

Ec(τ)——在0至τ期间改正的错误数。

2.基本假定

(1) 在类似的程序中，单位长度里的错误数ET/IT近似为常数。美国的一些统计数字表明，通常

0.5×10-2≤ET/IT≤2×10-2

(2) 失效率正比于软件中剩余的(潜藏的)错误数，而平均无故障时间MTTF与剩余的错误数成反比。

(3) 假设发现的每一个错误都立即正确地改正了(即调试过程没有引入新的错误)。因此，Ec(τ)=Ed(τ)。剩余的错误数为Er(τ)=ET -Ec(τ)，单位长度程序中剩余的错误数为εr(τ)=ET/IT - Ec(τ)/IT。

1. 估算平均无故障时间

平均无故障时间与单位长度程序中剩余的错误数成反比，公式：

4.符号

(1) 植入错误法

在测试之前由专人在程序中随机地植入一些错误，测试之后，根据测试小组发现的错误中原有的和植入的两种错误的比例，来估计程序中原有错误的总数ET。

假设人为地植入的错误数为Ns，经过一段时间的测试之后发现ns个植入的错误，此外还发现了n个原有的错误。如果可以认为测试方案发现植入错误和发现原有错误的能力相同，则能够估计出程序中原有错误的总数为

其中, 即是错误总数ET的估计值。

2) 分别测试法

为了随机地给一部分错误加标记，分别测试法使用两个测试员(或测试小组)，彼此独立地测试同一个程序的两个副本，把其中一个测试员发现的错误作为有标记的错误。具体做法是，在测试过程的早期阶段，由测试员甲和测试员乙分别测试同一个程序的两个副本，由另一名分析员分析他们的测试结果。用τ表示测试时间，假设

τ=0时错误总数为B0;

τ=τ1时测试员甲发现的错误数为B1;

τ=τ1时测试员乙发现的错误数为B2;

τ=τ1时两个测试员发现的相同错误数为bc。