软件质量保证与测试

Software Quality Assurance and Testing

第 4 章 白盒测试

4.5 程序变异测试和符号测试

金陵科技學院

程序变异测试：做一个类比

某医院新引进了一台检查某种疾病的仪器设备，设备安装好后，用这台设备对一批疑似病人进行了检查，检查结果是这些疑似病人都没有患病。此时，我们无法断定是这些人确实没有病，还是这台仪器有问题。有可能这些人中有人确实是患病者，只是检查结果有误，仪器没有把病人检查出来。

测试有问题 or 程序质量高

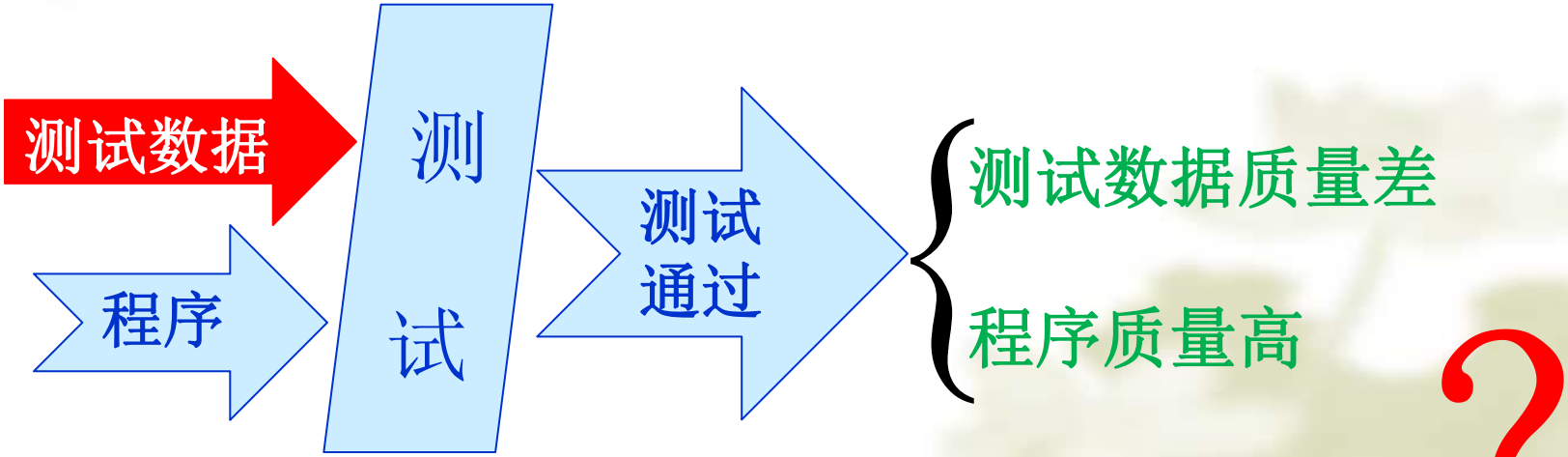
软件测试中也有类似情况。假设在对某个软件进行测试时，我们设计并执行了大量测试数据，但没有发现程序有什么问题，执行结果都是正确的。这时有两种可能：

一是这个软件确实质量很高，基本上没有什么问题；

二是我们设计的测试数据质量太差，发现不了程序中的问题。

测试有问题 or 程序质量高

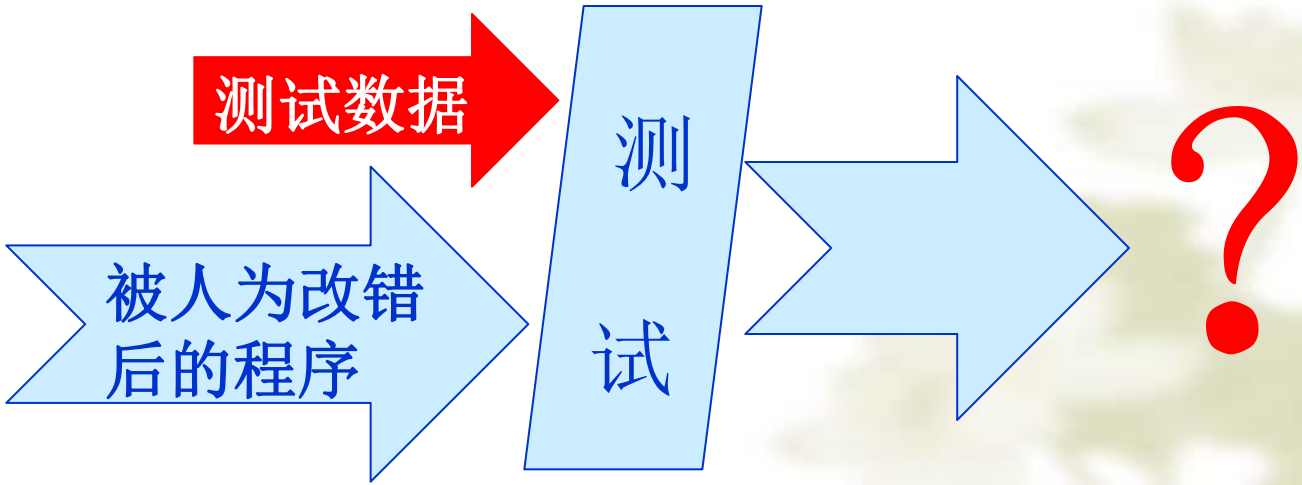
也就是说，采用大量测试数据对软件进行测试后，如果没有发现什么问题，那么既可能是程序质量高，也很可能是测试数据质量太差，发现不了问题！



那么到底是哪种情况呢？

程序变异测试

有一个办法可以来检验，那就是人为的按照某种规则把程序修改一下，让它有错误，然后再去执行前面的测试数据，看能不能发现我们修改程序后，人为植入的错误。



程序变异测试

如果能发现，则说明测试数据质量还是可以的，如果不能发现，则说明先前设计的测试数据质量确实不高，因为修改程序后植入的错误没有被发现！

这个例子可以帮助我们理解什么是变异测试，它有什么作用，当然这个例子只是变异测试的一种情况，变异测试的作用也不止这一种。

程序变异

变异测试中的程序变异是指：基于良好定义的变异操作，对程序进行修改，得到源程序的变异程序。而良好定义的变异操作可以是模拟典型的应用错误。

例如：模拟操作符使用错误，把大于等于写成小于等于，

或者是强制出现特定数据，以便对特定的代码或者特定的情况进行有效地测试，例如使得每个表达式都等于0，以测试某种特殊情况。

程序变异

程序变异通常只是一种轻微改变程序的操作。

例如，有程序段 P1：

```
if  (  x >= 60  )
        y = “合格”;
else
        y = “不合格” ;
```

可以用“>”来替换“>= ”，产生下面的变异程序 P2：

```
if  (  x > 60  )
        y = “合格”;
else
        y = “不合格” ;
```

变异测试

变异测试有时也叫做“变异分析”，是一种对测试数据集的有效性、充分性进行评估的技术，以便指导我们创建更有效的测试数据集。

变异测试

变异测试产生于20世纪70年代，最初是为了定位揭示软件测试中的不足，因为如果一个变异被引入，或者说一个已知的修改甚至是错误被植入到程序中，而测试结果不受影响，那么这就说明要么是变异代码没有被执行到，要么是程序的修改甚至是错误没有被测试工作检查出来。

如果变异代码没有被执行到，有可能是源程序中有过剩代码或者不可达代码，也可能是软件测试不充分，没有测试到这些代码。而如果变异代码被执行到了，但测试结果不受影响，那么就是测试无效，不能发现程序中的问题。

变异测试

事先被良好定义的变异操作可以称为变异算子。

给定一个程序P和一个测试数据集 T，通过变异算子为 P 产生一组变异体 M_i（必须是合乎语法的变更，变更后程序仍能执行），对 P 和 M 都使用 T 进行测试运行，如果某 M_i 在某个测试输入 t_j 上与 P 产生不同的结果，则该 M_i 被杀死；若某 M_i 在所有的测试数据集上都与 P 产生相同的结果，则称其为活的变异体。接下来对活的变异体进行分析，检查其是否等价于 P；对不等价于 P 的变异体 M 进行进一步的测试，直到充分性度量达到满意的程度。

变异测试

针对程序段 P1：

```
if  (  x >= 60  )
        y = “合格”;
else
        y = “不合格” ;
```

前面我们已经用“>”来替换“>= ”，产生了下面的变异程序 P2：

```
if  (  x > 60  )
        y = “合格”;
else
        y = “不合格” ;
```

变异测试

除了这种变异之外，还可以用“ = ”来替换“>= ”，产生下面的变异程序 P3 如下：

```
if  (  x = 60  )
        y = “合格”;
else
        y = “不合格” ;
```

变异测试

假设有一位测试员 A ，针对原来的程序段 P1 设计了测试数据集 T1，包括测试数据：

x = 70 和 x = 50

那么把这个测试数据集用于变异程序 P2 时，是发现不了问题的，两个测试数据都能得到正确的结果，这就可以提醒测试人员，还需要增加测试用例，x=60 才行。

变异测试

假设有一位测试员 B ，针对原来的程序段 P1 设计了测试数据集 T2，包括测试数据：

x = 60 和 x = 50

那么把这个测试数据集用于变异程序 P3 时，也是发现不了问题的，两个测试数据也都能得到正确的结果，这就可以提醒测试人员，还需要增加测试用例，例如 x=80 才行。

变异测试的优点

程序变异测试方法是一种错误驱动测试。该方法通常是针对某类特定的程序错误。经过多年的测试理论研究和软件测试的实践，人们逐渐发现要想找出程序中所有的错误几乎是不可能的。比较现实的解决办法是将错误的搜索范围尽可能地缩小，以利于专门测试某类错误是否存在。

变异测试的优点

从软件测试的角度来说，变异测试可以帮助测试人员发现测试工作中的不足，然后进一步提高测试数据集的覆盖度和有效性，改进和优化测试数据集。

另外变异测试还可用于在细节方面改进程序源代码。

变异测试的缺点

如果要让变异测试针对各种情况，则必须引入大量的变异，这将导致会有数量极大的程序变异体被编译、执行和测试，这将耗费大量的测试成本，变异测试的成本问题，阻碍了它成为一种软件测试的基本和常用方法。

另外变异测试的前提是需要有测试数据集 T，而这个测试数据集 T一般是采用其他方法设计出来的，所以说变异测试一般并不能单独使用，而需要与传统的其它测试方法技术相结合。

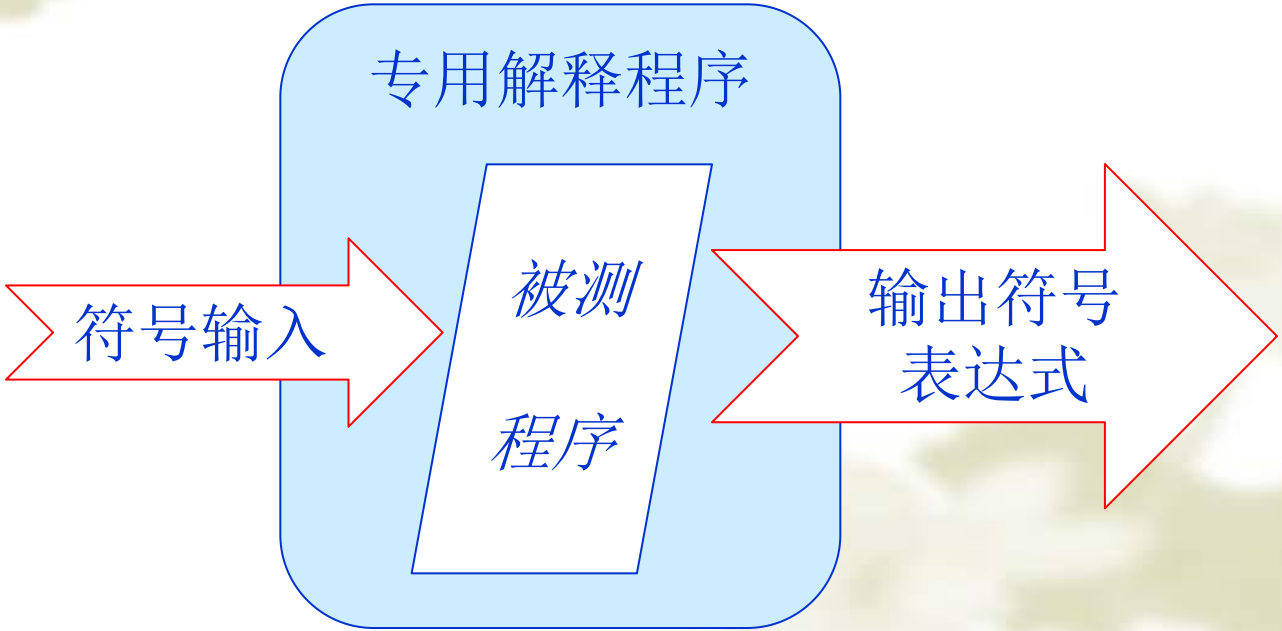
符号测试

下面我们来学习符号测试。符号测试的基本思想是允许程序的输入不仅是具体的数值数据，而且可以是符号值，这一方法也因此而得名。

符号执行法是一种介于程序测试用例执行与程序正确性证明之间的方法。它使用一个专用的程序，对输入的源程序进行解释。在解释执行时，允许程序的输入不仅仅是具体的数值数据，而且包括符号值，符号值可以是基本的符号变量，也可以是符号变量的表达式。

符号测试

符号测试的执行结果，是包含输入符号的表达式。



符号测试

符号测试执行得到的结果，可以有两个用途：

一是可以检查程序的执行结果是否符合程序的规格或者是预期的目的；

二是通过程序的符号执行结果，可以分析程序的执行路径，为进一步自动生成测试数据提供条件。

符号测试

下面我们来看两个简单的例子，以帮助我们理解符号测试。

设有一段程序，功能是计算两个数的和，如果要把两个数相加所有可能的情况如 1+1 ，1+2，2+1，.....等等，都输入进去测试一次，这是不可能做到的，也是没有必要的。于是我们会想，是不是可以输入两个符号，A 和 B，只要执行结果是 A + B，那么程序就是正确的。

这就是通过符号测试来检查程序执行结果是否符合程序的规格或者是预期的目的。当然这一般只适用于简单的程序。

符号测试

再来看另外一个例子，设有程序段 P1：

```
if  （  x >= 60  ）
        y = “合格”；
else
        y = “不合格”  ；
```

对其进行测试时，如果要把 x 的所有取值如 x= 10，15，20，80.5，.....等等，都输入进去测试一次，测试工作量还是很大的。

此时可以采用符号测试，输入符号 C。

符号测试

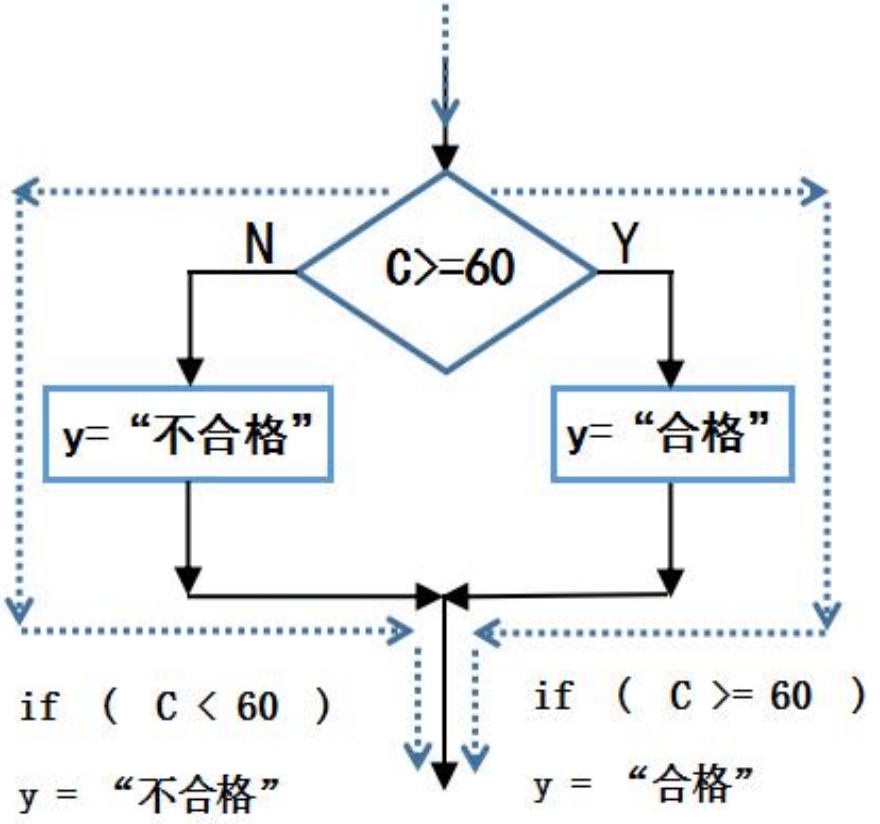
对程序段 P1，输入符号 C 后的执行结果，一般是如下形式的符号表达式组：

$$\begin{cases} \text{“if (C >= 60) y = “合格””} \\ \text{“if (C < 60) y = “不合格””} \end{cases}$$

通过符号测试执行的这一结果，我们可以分析出程序有两条执行路径，两条执行路径分叉的依据是输入数据是否 >= 60，这样我们就可以针对这两条路径设计测试数据，如 70 和 50。而且，通过一定的技术手段，这样的测试数据可以自动生成。

符号测试

符号测试中的解释程序需要在被测试程序的判定点计算谓词。例如，对程序段 P1 进行符号测试时判断输入数据 C是否 >=60。很显然，一个 IF 语句就会形成两个执行分支。



符号测试

一个判断语句 if……then…else 的两个分支在一般情况下需要进行并行计算。语法路径的分支形成一棵“执行树”，树中每一个结点都是一个表示执行到该结点时累加判定的谓词，也就是包含输入符号、判断和运算的表达式。

符号测试

一旦解释程序对被测源程序的每一条语法路径都进行了符号计算，就会对每一条路径给出一组输出，它是用输入符号，再加上遍历这条路径所必须满足的条件的谓词组，这两者的符号形式表示的。

实际上，这种输出包含了程序功能的定义。在理想情形下，这种输出可以自动地与被测程序所要具备的功能，也就是它的程序规格，进行比较。否则可用手工进行比较。

符号测试

由于语法路径的数目可能很大，再加上其中有许多是不可达路径，这时需要对执行树进行修剪。但是修剪时必须特别小心，不要把“重要”路径无意中修剪掉。另外，还有一个问题：如果对象源程序中包含有循环，而循环的结束取决于输入的值，那么执行树就会是无穷的，这时，必须加以人工干预，进行某种形式的动态修剪，以保证解释执行是可以终止的，而不会无限进行下去。

符号测试

符号执行更有用的一个结果是用于产生测试数据。符号执行的各种语法路径输出的累加谓词组(只要它是可解的)定义了一组等价类，每一等价类又定义了能够经过该路径的输入应当满足的条件，因此可依据这种信息来选择测试数据。寻找好的测试数据就等于寻找语义(即可达)路径。

符号执行方法还可以度量测试覆盖程度。如果路径谓词的析取值为真(true)，则该测试用例的集合就“覆盖”了源程序。如果不是这样，该析取值的取假(false)，表示源程序中还有没有被 测试到的区域。

本节内容就讲到这里，谢谢，再见！

金陵科技學院