**[LilianChen](http://www.cnblogs.com/LilianChen/)**

[测试的艺术：测试用例的设计](http://www.cnblogs.com/LilianChen/p/4053063.html)

由于时间和成本的约束，软件测试的最关键问题是：

**在所有可能的测试用例中，哪个子集最有可能发现最多的错误**

**测试方法：**

黑盒测试

* **等价类划分（Equivalence Partitioning）**

1. 严格控制测试用例的增加，减少为达到“合理测试”的某些既定目标而必须设计的其他测试用例的数量

2. 它覆盖了大部分其他可能的测试用例。

划分了等价类后，就可以说，如果对该集合中某个元素所进行的测试没有发现错误的话，那么对该集合中其他元素所进行的测试也不大可能会发现错误。

使用等价类划分方法设计测试用例主要有两个步骤：（1）确定等价类； （2）生成测试用例

（1）确定等价类

外部条件，有效等价类（代表对程序的有效输入），无效等价类（其他任何可能的输入条件-即不正确的输入值）

确定等价类大体上是一个启发式的过程，一些指导原则：

* 如果输入条件规定了一个取值范围（eg. “数量可以是从1到99”），那么就应确定出一个有效等价类（1<数量<99），以及两个无效等价类（数量<1, 数量>99）
* 如果输入条件规定了取值个数（eg. “汽车可登记一至六名车主”），那么就应确定出一个有效等价类和两个无效等价类（没有车主，或车主对于六个）
* 如果输入条件规定了一个输入值的集合，而且有理由认为程序会对每个值进行不同的处理（eg. “交通工具的类型必须是公共汽车、卡车、出租车、火车或摩托车”），那么就应为每个输入值确定一个有效等价类和一个无效等价类（eg. 拖车）
* 如果存在输入条件规定了“必须是”的情况(eg. “标识符的第一个字符必须是字母”)， 那么就应确定一个有效等价类（首字符是字母）和一个无效等价类（首字符不是字母）

（2）生成测试用例

使用等价类来生成测试用例的过程：

* 为每个等价类设置一个不同的编号
* 编写新的测试用例，尽可能多地覆盖那些尚未被涵盖的有效等价类，知道所有的有效等价类都被测试用例所覆盖
* 编写新的用例，覆盖一个且仅一个尚未被覆盖的无效等价类，直到所有的无效等价类都被测试用例覆盖： 用单个测试用例覆盖无效等价类，是因为某些特定的输入错误检查可能会屏蔽或取代其他输入错误检查。

* **边界值分析(Bondary-Value Analysis)**

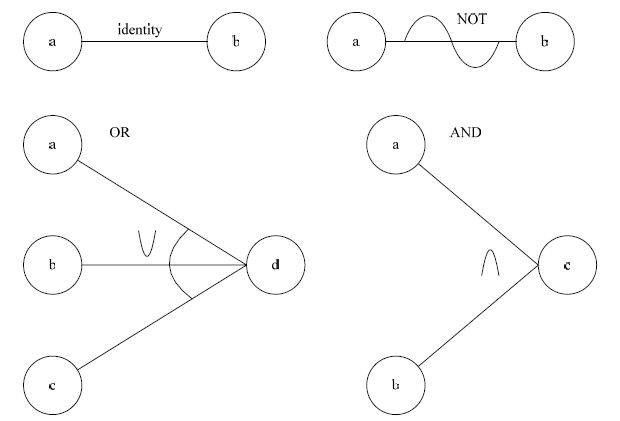
所谓边界条件，是指输入和输出等价类中那些恰好处于边界、或超过边界、或在边界以下的状态。

* **因果图分析（Cause-Effect Graphing）**

　　生成测试用例时采用的过程：

* + 将规格说明分解为可执行的片段
  + 确定规格说明中的因果关系。所谓因，是指一个明确的输入条件或输入条件的等价类。所谓果，是指一个输出条件或系统转换。
  + 分析规格说明的语义内容，并将其转换为连接因果关系的布尔图。这就是所谓的因果图
  + 给图加上注解符号，说明由于语法或环境的限制而不能联系起来的因和果
  + 通过仔细地跟踪图中的状态变化情况，将因果图转换成一个有限项的判定表。表中的每一列代表一个测试用例
  + 将判定表中的列转换成测试用例

　　因果图中的基本符号：



设想一下，每个结点的值为0或为1，0表示“不存在”状态， 1 表示“存在” 状态。identity函数表示，如果a等于1，则b也是1，否则b为0. not函数表示如果a等于1，则b为0，否则b为1.Or函数表示如果a或b或c等于1，则d为1，否则d为0。 and函数表示如果a和b都等于1，则c为1，否则c为0.

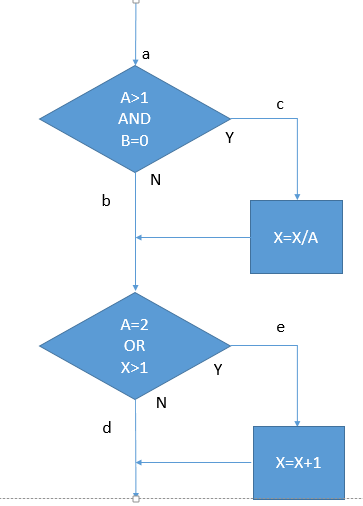
* **错误猜测（Error Guessing）**

**白盒测试（White-Box Testing）**

* **语句覆盖**
  + 度量被测代码中每个可执行语句是否被执行到了。
  + int foo(int x, int y)
  + {
  + return x / y;
  + }

测试用例，x=10, y=5.

* + 测试人员的测试结果会告诉你，代码覆盖率达到了100%，并且所有测试案例都通过了。然而遗憾的是，却没有发现最简单的bug，比如，让y=0时，会抛出一个除零异常。
* **判定覆盖**
  + 该准则要求必须编写足够的测试用例，使得每一个判断都至少有一个为“真”和为“假”的输出结果， 并且每条语句都至少被执行一次。换句话说，即每个判断都必须有“是”和“否”的结果，且每个入口点都必须至少被调用一次。
* **条件覆盖**
  + 在条件覆盖情况下，要编写足够的测试用例以确保将一个判断中的每个条件的所有可能的结果至少执行一次，且每个入口点都必须至少被调用一次。
* **判定/条件覆盖**
  + 将一个判断中的每个条件的所有可能的结果至少执行一次，将每个判断的每个条件的所有可能的结果至少执行一次，将每个判断的所有可能的结果至少执行一次，将每个入口点都至少调用一次。
* **多重条件覆盖**
  + 该准则要求编写足够多的测试用例，将每个判定中的所有可能的条件结果的组合，以及所有入口点都至少执行一次。
  + 其中的原因是“与”和“或”表达式中某些条件的结果可能会屏蔽或阻碍其他条件的判读。举例来说，如果“与”表达式中有个条件为“假”，那么就无须计算该表达式中的后续条件。



[复制代码](javascript:void(0);)

public void foo(int a, int b, int x)

{

if (a > 1 && b == 0)

{

x = x / a;

}

if (a == 2 || x > 1)

{

x = x + 1;

}

}

[复制代码](javascript:void(0);)

* 语句覆盖： ace - A=2, B=0, X=3
* 判定覆盖：acd, abe - A=3, B=0, X=3; A=2, B=1, X=1
* 条件覆盖：abe - A=2, B=0, X=3; A=1, B=1, X=1
* 多重条件覆盖：测试用例必须覆盖一下8中组合
  + A>1, B=0
  + A>1, B<>0
  + A<=1, B=0
  + A<=1, B<>0
  + A=2, X>1
  + A=2, X<=1
  + A<>2, X>1
  + A<>2, X<=1
    - A=3, B=0, X=4
    - A=2, B=1, X=1
    - A=1, B=0, X=2
    - A=1, B=1, X=1

**测试策略：**

1. 如果规格说明中包含输入条件组合的情况，应首先使用因果图分析方法

2. 在任何情况下都应使用边界值分析方法。对输入和输出边界进行的分析

3. 应为输入和输出确定有效和无效等价类，在必要情况下对上面确认的测试用例进行补充

4. 使用错误猜测技术增加更多的测试用例

5. 针对上述测试用例集检查程序的逻辑结构。应使用判定覆盖、条件覆盖、判定/条件覆盖或多重条件覆盖准则。