第6章　软件测试（上）

- 软件测试基础
- 白盒测试
- 黑盒测试
- 白盒测试和黑盒测试的比较
- 测试用例的有效性

软件测试基础

软件测试背景

例一：在某撑杆跳横杆自动调节系统中，如果运动员撑杆跳的高度达到6米，则横杆自动升高5厘米。

```
if (Height == 6)
{
    // 横杆自动升高5厘米
    Height  *= 1.05;
}
```

例二：在某嵌入式控制系统中，需反复检测各标志位的状况。

```
int i;
for (i = 0; i < 100000; i++)
{
    //  检测各标志位
}
```

软件测试背景

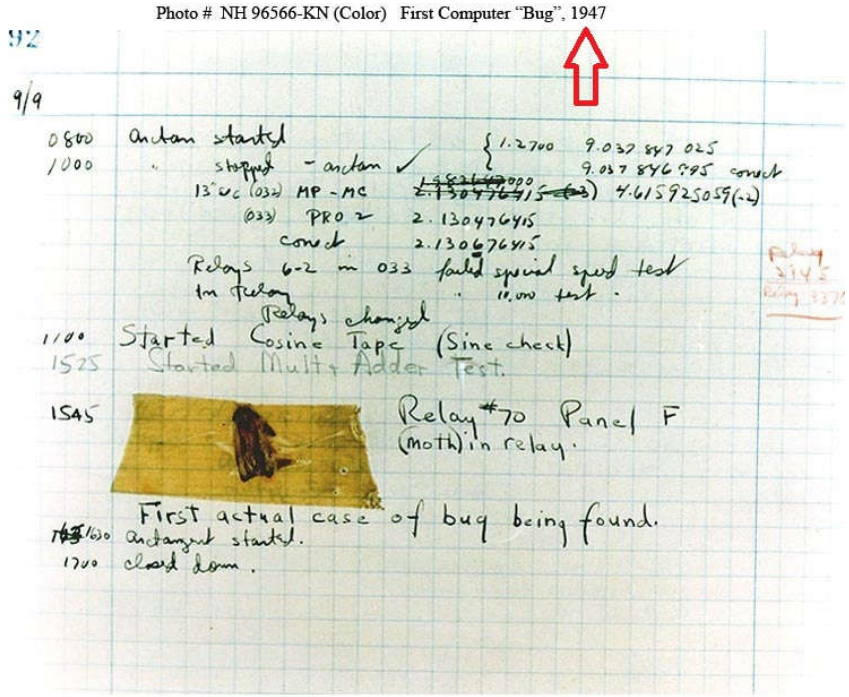
例3：买了一部新手机，（在7日内）如何进行有效测试？

- 检查手机屏幕、外观是否有损坏？是否能正常开机、启动、关机？安装的APP，是否能正常使用？是否能正常接打电话？能够正常上网？
- 能否连续长时间充电？ 能够长时间游戏？追剧？还是上乐学？低电量时手机性能怎么样？
- 同学，你还能想到什么.....

软件测试基础

软件测试——找“bug”的由来

Grace Murray Hoper



Lieutenant Grace Hopper is part of a team that finds a moth that is bugging up the Mark II Aiken Relay Calculator at Harvard. After debugging the system, the moth is affixed to the computer log, where Hopper notes: "First actual case of bug being found.“

——https://www.history.navy.mil/today-in-history/september-9.html

软件测试基础

对于软件测试的定义，有如下不同的描述：

- **IEEE（1983）**：使用人工或自动运行测试系统的过程，其目的在于检验系统是否满足用户需求，或找出预期结果与实际运行结果间的差别，发现程序错误。
- **Glen Myers**：软件测试为了发现错误而执行程序的过程。
- 从软件质量和可靠性角度理解，软件测试是为保证软件质量、提高软件可靠性的活动，它应用测试理论和技术，发现程序中的错误和缺陷而实施的过程。

软件测试基础

软件测试过程主要包括测试对象和测试方法两部分内容。

测试对象分为程序代码和文档。

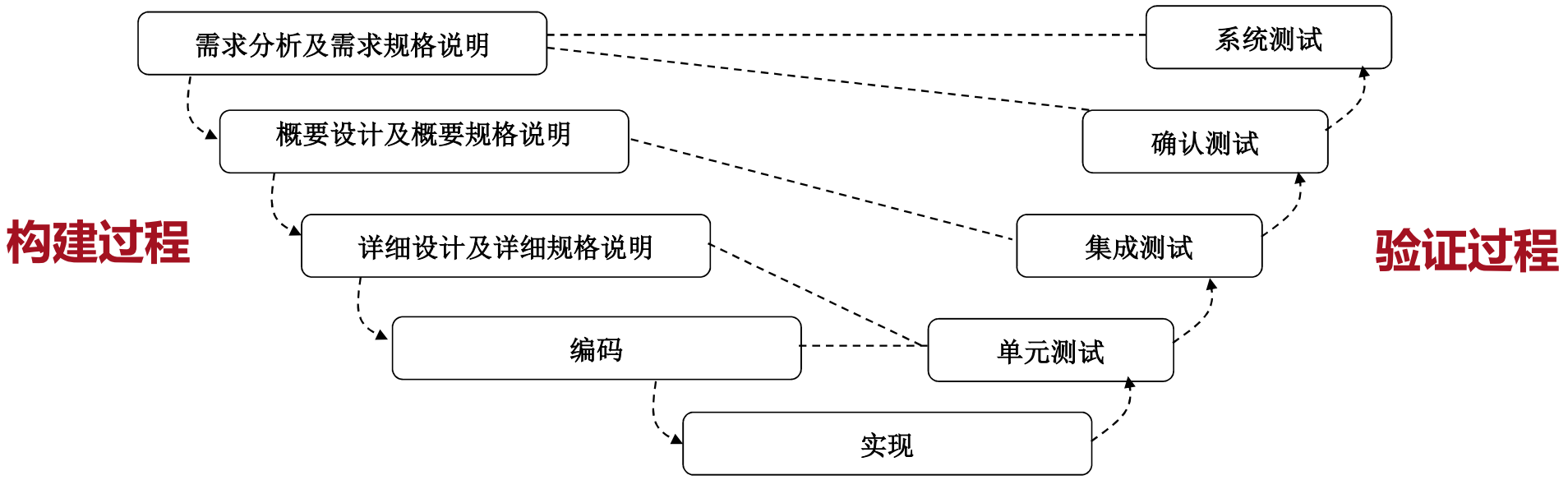
文档分为技术文档和用户文档。

- 对技术文档，检查设计方案是否符合需求；
- 对技术文档，检查设计方案是否正确；
- 对用户手册，提供的系统安装过程说明是否详尽；
- 对用户手册，提供的系统操作方法、过程是否正确；
- 对用户手册，提供的示例数据是否准确；
- 对用户手册，提供的信息是否完整、详尽且无二义性.

软件测试基础

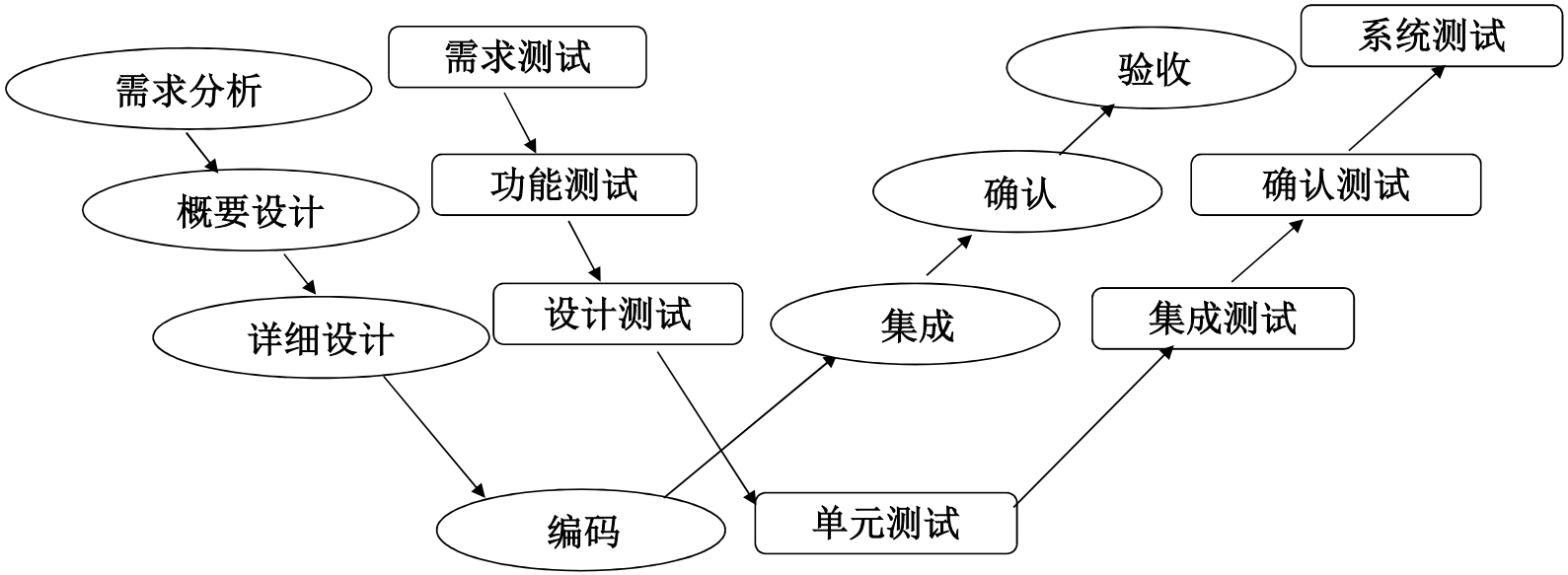
1. 软件测试过程模型——V模型

V模型的重要价值在于，它定义了软件测试如何与软件工程各阶段相融合，它清楚地描述了各级别软件测试与软件开发各阶段的对应关系。



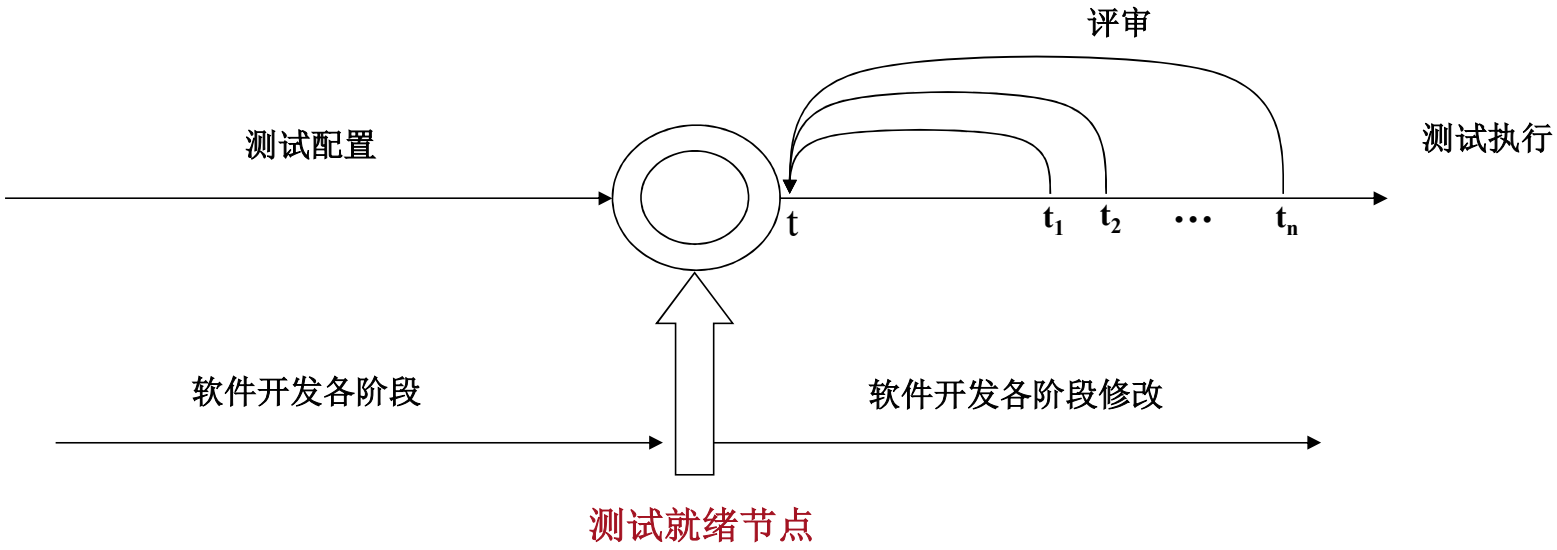
2.软件测试过程模型——W模型

W模型的重要贡献在于，明确软件开发各阶段都要进行测试，而不仅仅是在编码结束后才开始。这样，测试的对象不仅是代码，还可以是文档（需求规格说明、设计规格说明等）。



3. 软件测试过程模型——H模型

软件测试的H模型是对W模型在更高层次上的线性抽象。它明确表示，在任何一个开发流程，只要有必要，并且测试配置已准备就绪，就能进行测试活动。



软件测试基础

基本的软件测试技术

- 静态测试与动态测试
- 白盒测试与黑盒测试
- 测试策略

软件测试基础

软件测试技术分类（一）——静态测试

静态测试的测试对象包括源程序和文档。项目开发过程中产生大量的规格说明，对这些规格说明的技术审查和管理复审，以及对文档的测试数据都属于静态测试。

静态测试通过人工分析或程序正确性证明的方式来确认程序正确性。

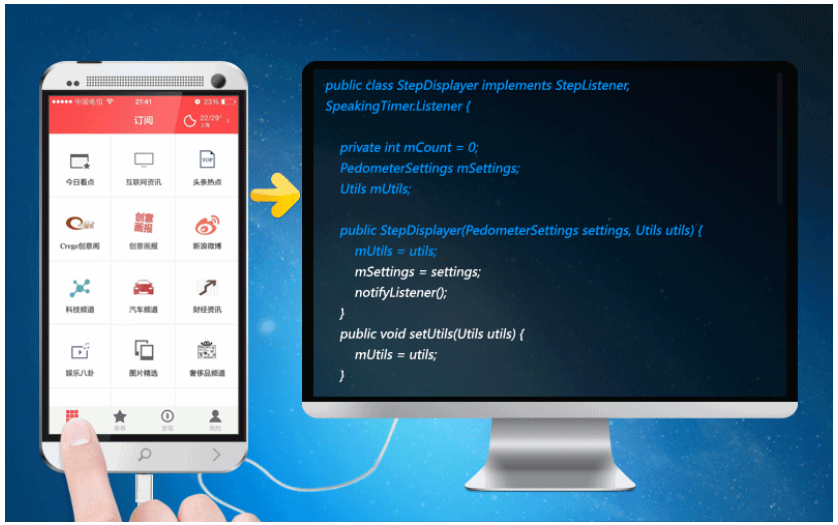
非正式 ——— 正式

互审　　走查　　审查会议

软件测试基础

软件测试技术分类（一）——动态测试

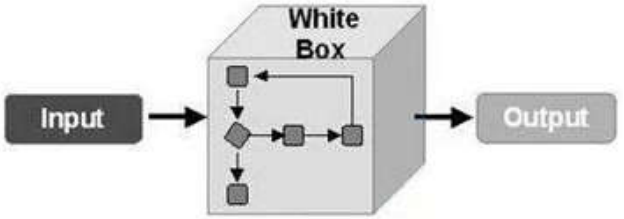
动态测试的测试对象针对源程序。就源程序来讲，静态测试是指不运行程序就找出程序中存在的错误，动态测试是通过运行程序而发现存在的错误和问题。



软件测试基础

软件测试技术分类（二）——白盒测试

白盒测试又称为结构测试、基于覆盖的测试。它是针对模块内部逻辑结构进行的测试。它面对程序内部的实现细节，分别对语句、条件、条件组合、循环等控制结构、异常、错误处理等特殊流程设计测试用例。



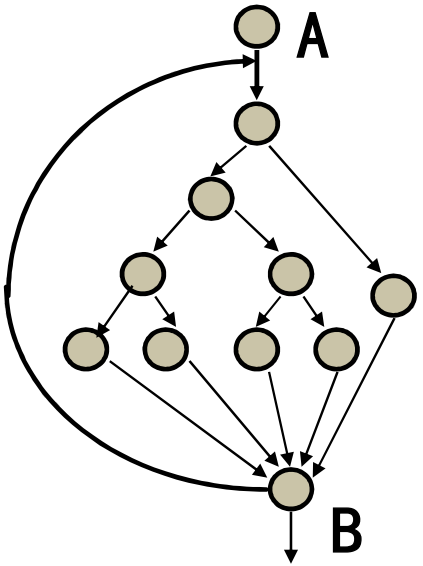
- 白盒测试中的穷尽测试

例：含5个分支，循环次数≤20，从A到B的可能路径：

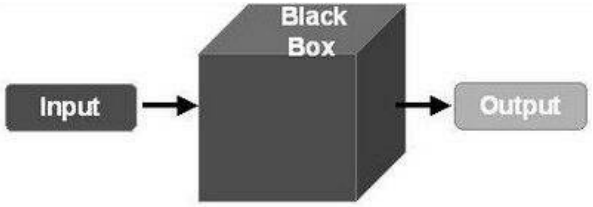
$5^1 + 5^2 + \cdots + 5^{19} + 5^{20} \approx 1.2 \times 10^{14}$

执行时间：设测试一次需要2ms，

则穷尽测试需要约3万年



软件测试技术分类（二）——黑盒测试

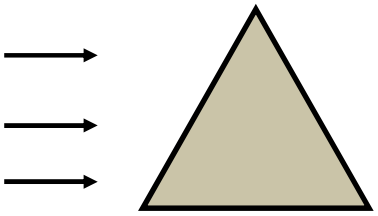


黑盒测试是把模块作为一个整体进行测试。它不关心程序逻辑实现的具体细节，而是关注模块的输入（接口）、输出（运行结果）。因而它测试的是模块功能是否符合设计，运行时是否能（被）正确调用。

- 黑盒测试中的穷尽测试

例:输入三条边长用黑盒测试中的穷尽测试，可采用的测试用例数(设字长32位)：

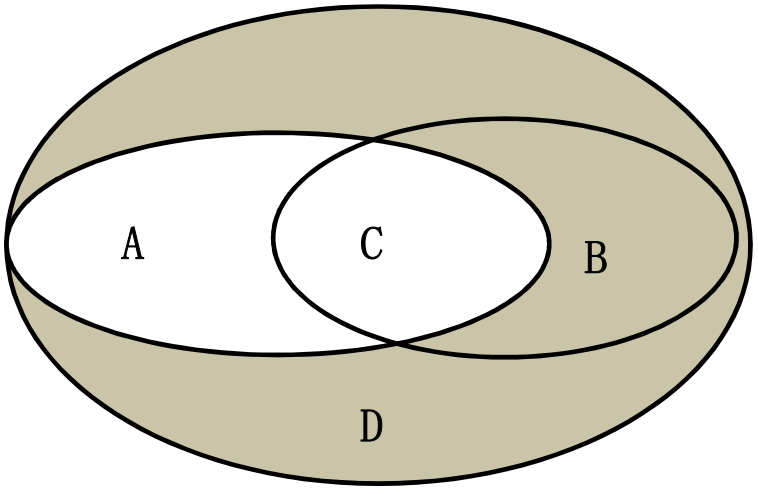
$$2^{32} \times 2^{32} \times 2^{32} \approx 2^{96}$$



执行时间：设测试一次需1毫秒，共需约2^{61}年。

软件测试基础

在实际测试过程中，无论是黑盒测试还是白盒测试都不能进行穷尽测试，所以软件测试不可能发现程序中存在的所有错误。为此，需精心组织测试方案、设计测试用例，力争用尽可能少的测试用例，发现尽可能多的错误。



黑盒测试与白盒测试能发现错误的关系

A 只能用黑盒测试发现的错误；

B 只能用白盒测试发现的错误；

C 两种方法都能发现的错误；

D 两种方法都不能发现的错误。

软件测试基础

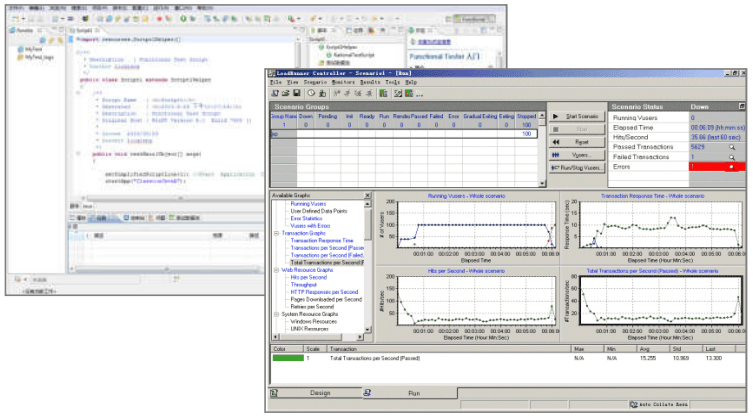
软件测试的局限性

测试的不彻底性

- 测试只能说明程序中错误的存在，但不能说明错误不存在。
- 经过测试后的软件不能保证没有缺陷和错误。

测试的不完备性

- 测试无法覆盖到每个应该测试的内容。
- 不可能测试到软件的全部输入与响应。
- 不可能测试到全部的程序分支的执行路径。



测试作用的间接性

- 测试不能直接提高软件质量，软件质量的提高要依靠开发。
- 测试通过早期发现缺陷并督促修正缺陷来间接地提高软件质量。

白盒测试技术——逻辑覆盖

- 逻辑覆盖准则

白盒测试技术——逻辑覆盖

- 逻辑覆盖准则

覆盖强度	名 称	覆盖标准
1	语句覆盖	程序中的每条语句都至少执行一次。
2	判定覆盖	程序中的所有判定的每个分支都至少执行一次。
3	条件覆盖	程序中每个判定的各个子关系表达式的取值都至少执行一次。
4	判定/条件覆盖	程序中的所有判定的每个分支都至少执行一次。同时，每个判定的各个子关系表达式的取值都至少执行一次。
5	条件组合覆盖	程序中每个判定中的各子关系表达式的取值组合都至少执行一次。

白盒测试

练习：

```
void Example(double x, double y, double z)
{
    if ((y > 1) && (z == 0))    x /= y;
    if ((y == 2) || (x == 1))   x++;
}
```

请分别给出满足语句覆盖、条件覆盖和条件组合覆盖的测试用例。

白盒测试

白盒测试——循环测试

循环测试是三种基本控制结构之一，循环测试的目的是检查循环结构的有效性。循环分为简单循环、嵌套循环、并列循环和非结构循环四类，如图所示。

对于最多为n次的简单循环，应做下列测试：

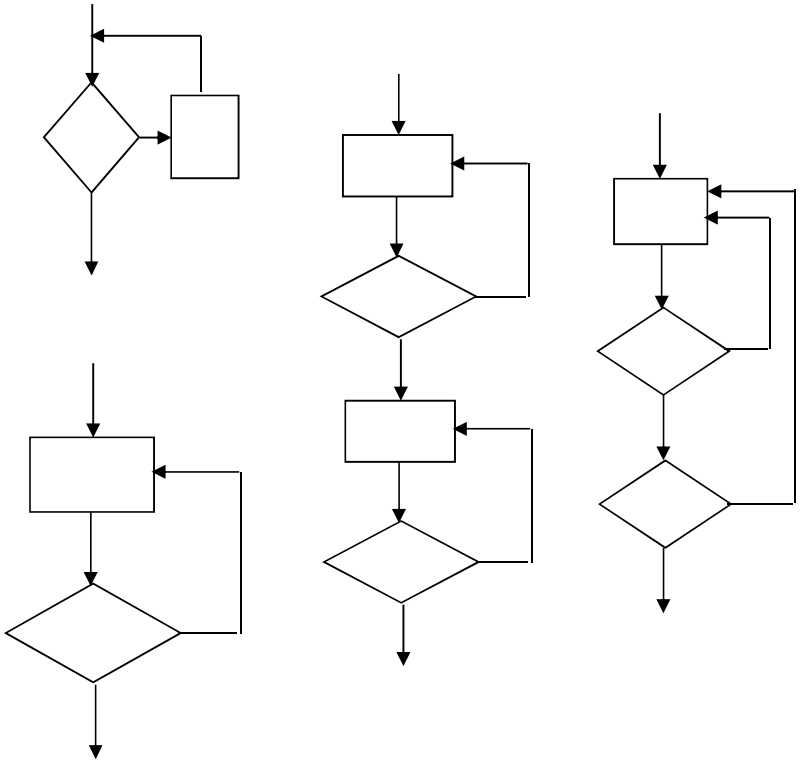
（1）完全跳过循环

（2）仅循环一次；

（3）循环两次；

（4）循环m次，m＜n；

（5）分别循环（n-1)次、n次、n＋1次。

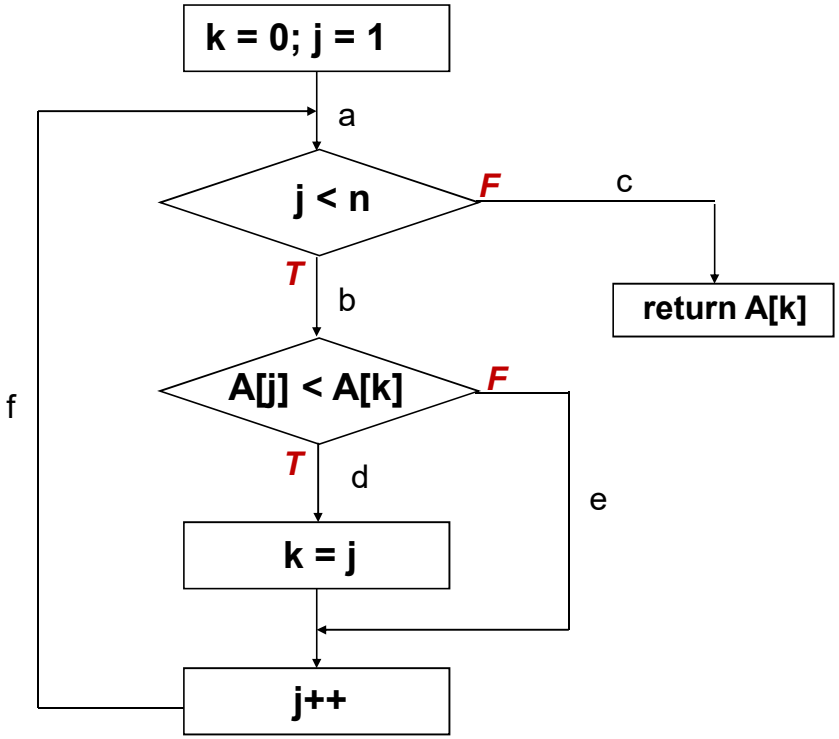


简单循环　　顺序循环　　嵌套循环

白盒测试——单循环测试　练习

问题：求整型数组A中元素的最小值。

```
k = i = 0;
for (j = i+1; j < n; j++)
{
        if (A[j] < A[k])
        {
            k = j;
        }
}
return  A[k];
```



白盒测试

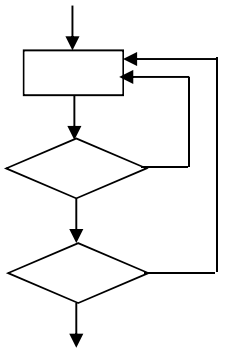
对于嵌套循环若生搬硬套简单循环的测试策略，可能使测试次数成几何级数增长。因此需要在确保循环测试有效性的前提下，采用如下的测试方法以减少测试次数：

（1）从最内层循环开始测试，此时所有外层循环都取最小值，内层循环按简单循环的测试策略测试。

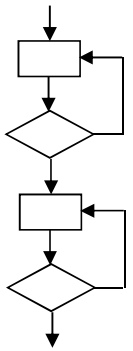
（2）由里向外，回退到上一层循环测试，这层循环的所有外层循环仍取最小值，由该层循环嵌套的那些循环取一些典型值。

（3）继续向外扩展，直至所有循环测试完毕。

（4）对于顺序循环分两种情况，若两个循环完全独立，采用简单循环的测试策略，反之，若第一循环的计数器用作第二循环的初值，即两循环不独立，需借鉴嵌套循环的测试策略。



嵌套循环



顺序循环

白盒测试

白盒测试——嵌套循环测试　练习

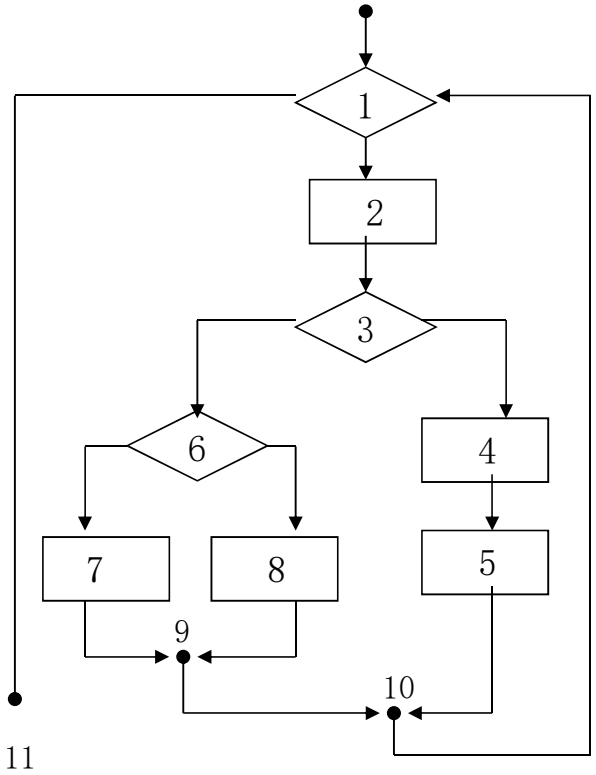
```
for (i = 0; i <= num; i++)
{
    while (j > 0)
    {
        j--;
    }
}
```

思考：

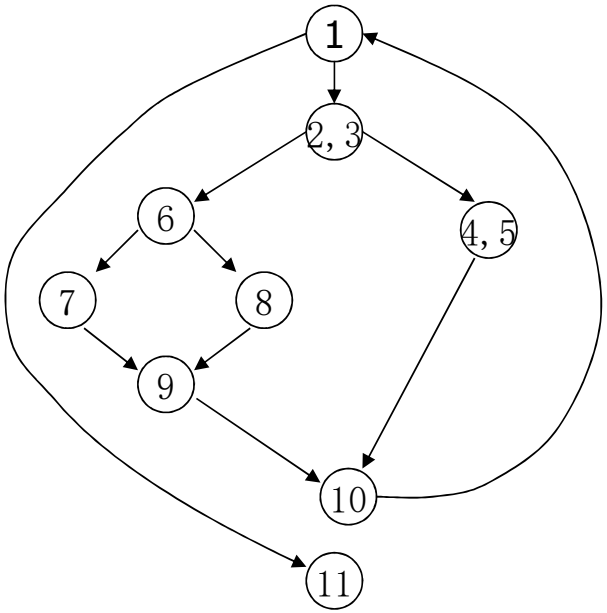
```
for (i = 0; i <= num1; i++)
{
    for (k = i; k <= num2; k++)
    {
        while (j > 0)
        {
            j--;
        }
    }
}
```

白盒测试

白盒测试——路径测试



待测试程序



用流图表示的待测试程序

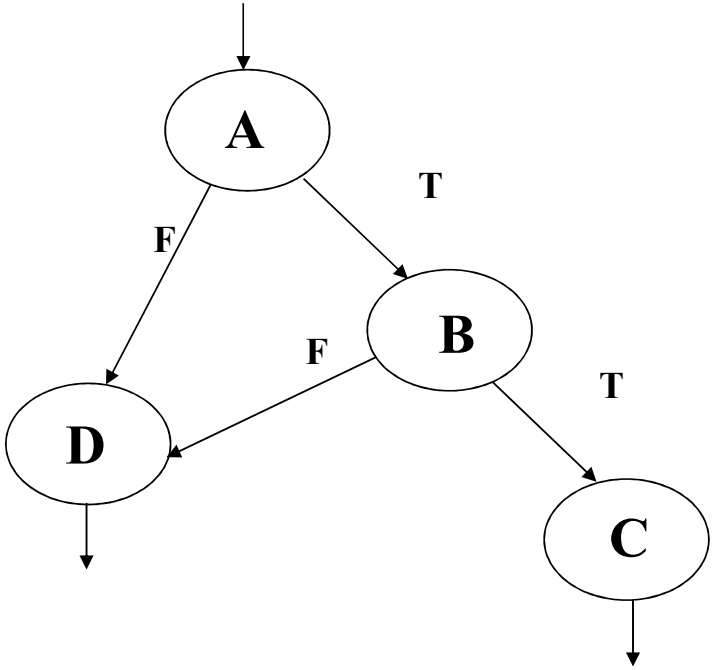
白盒测试

流图中“与”节点的逻辑表示

if (A and B)

then C

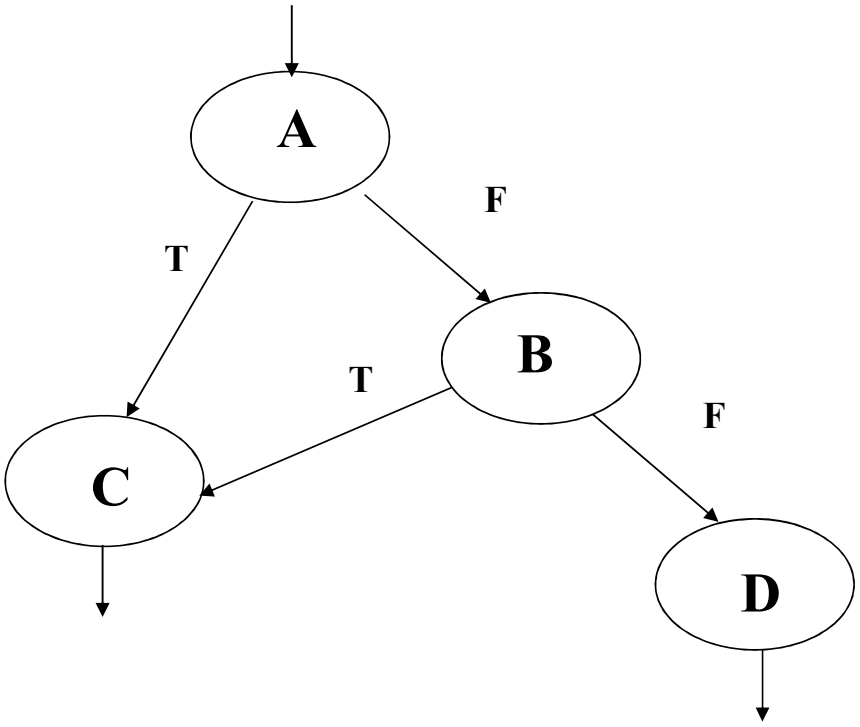
else D



白盒测试

流图中“或”节点的逻辑表示

```
if  (A or B)
   then  C
   else  D
```



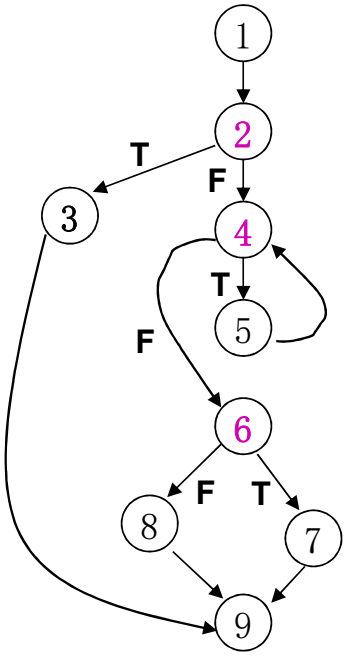
白盒测试

路径测试练习

```
double GetMean(const string& path) {
    int score, sum = 0, num = 0;          } // (1)
    ifstream ScoreFile(path);

    if (!ScoreFile)   // (2)
       throw("File is NOT Found!");   // (3)
    // Calculate the total score
    while (ScoreFile >> score) // (4)
    {
        sum += score;                     } // (5)
        num++;
    }
    // Compute the mean and print the result
    if (num > 0) // (6)
        return (double)sum / num; // (7)
    else
        throw("No scores found in file"); // (8)
} // (9)
```

1. 画出流图



2. 计算独立路径数:

V(G) = 封闭区域数目

V(G) = 边数 - 节点数 + 2

V(G) = 判断节点数 + 1

3. 给出独立路径及对应的测试用例

Path1: (1) (2) (3) (9)

Path2: (1) (2) (4) (6) (7) (9)

Path3: (1) (2) (4) (6) (8) (9)

Path4: (1) (2) (4) (5) (6) (8) (9)

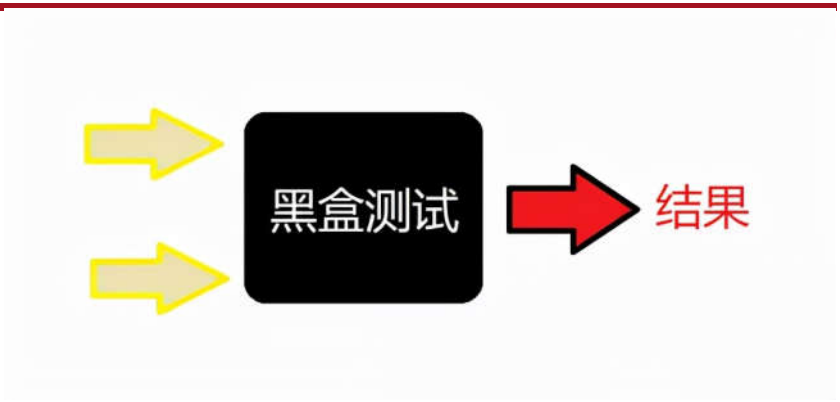
白盒测试

白盒测试方法的比较

方法	语句覆盖	判定覆盖	条件覆盖	条件/判定覆盖	条件组合覆盖	路径覆盖
优点	最基本、简单的覆盖形式。	简单，无需细分每个条件。	对容易出错的条件进行的测试。	兼顾判定和判定中各条件的取值判断。	对程序进行较彻底的测试，覆盖面广。	清晰、测试用例有效。
缺点	对于判定、条件、路径都没有涉及，是粗粒度的覆盖。	往往大部分的判定语句是由多个逻辑条件组合而成，仅仅判断其组合条件的结果，而忽视每个条件的取值情况，必然会遗漏部分测试场景。	测试用例较多，同时也不能完全涵盖判定覆盖。	测试用例较多，同时也不能完全涵盖路径覆盖。	设计大量复杂的测试用例，工作量较大。	不能替代条件组合覆盖。

黑盒测试

黑盒测试又称功能测试或行为测试，它主要根据设计说明中的功能设计来测试程序能否按预期实现。



黑盒测试的目的是尽量发现系统功能中的错误。常见的系统功能错误有以下几类：

- 功能不正确或不完整；
- 界面或接口错误；
- 数据结构错误；
- 访问外部数据库错误；
- 性能不满足需求；
- 初始化或终止系统时的错误。

黑盒测试

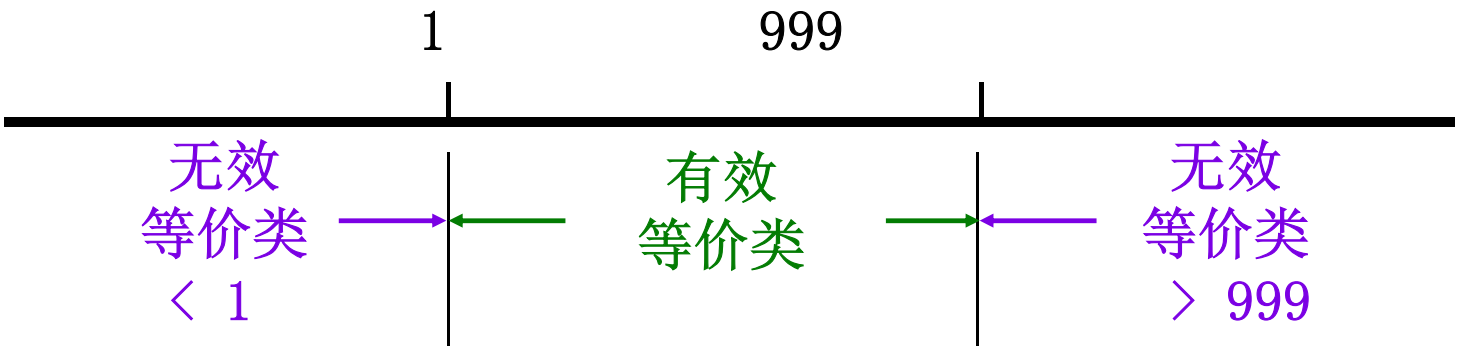
黑盒测试方法

黑盒测试——划分等价类的规则

对于常见的数据分析，如特殊数值、区间值、布尔值等，结合各类经验，有以下原则辅助确定等价类：

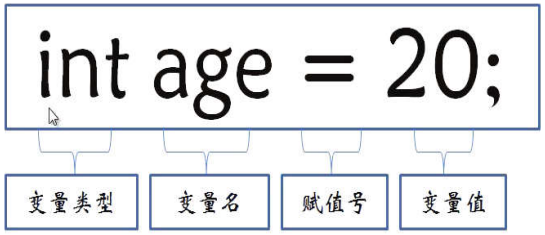
⑴ 如果定义了输入数据的取值范围（如**[a. b]**），则可划分一个有效等价类（**[a. b]**间的数据集）和两个无效等价类（**-∞, a**）以及（**b, +∞**）。

例 输入条件规定：项数可从1到999



黑盒测试

黑盒测试——划分等价类的规则



⑵ 如果规定了输入数据的个数（如**N**个），则可以划分出一个有效等价类（**1**～**N**之间）和两个无效等价类（**0**个）或（**N+M**个数据）。

例如，在C语言中，对变量标识符规定“以字母或者下划线开头的字符串”。那么所有以字母开头的串构成一个有效等价类、以下划线开头的串构成另一个有效等价类，而不在这两个等价类集合中（不以字母或下划线开头）的串归于无效等价类。

黑盒测试

黑盒测试——划分等价类的规则

(3) 如果规定输入数据是特殊值，则特殊值集合是有效等价类，其余取值构成一个无效等价类。

(4) 如果输入数据是布尔量，则可划分出一个有效等价类和一个无效等价类。

(5) 如果定义了输入的数据规则，则可划分出一个符合规则的有效等价类和一个违反规则的无效等价类。

(6) 如果输入的数据是整型，则可划分负数、零和正数三个有效等价类。

(7) 对于上述各自划分的有效等价类和无效等价类，可以根据不同角度、规则、程序处理方式等各方面入手，再细分为若干有效或无效的等价子类。

黑盒测试

黑盒测试——划分等价类的步骤

第一步：根据输入数据，划分待测问题的等价类，并对每个等价类进行编号；

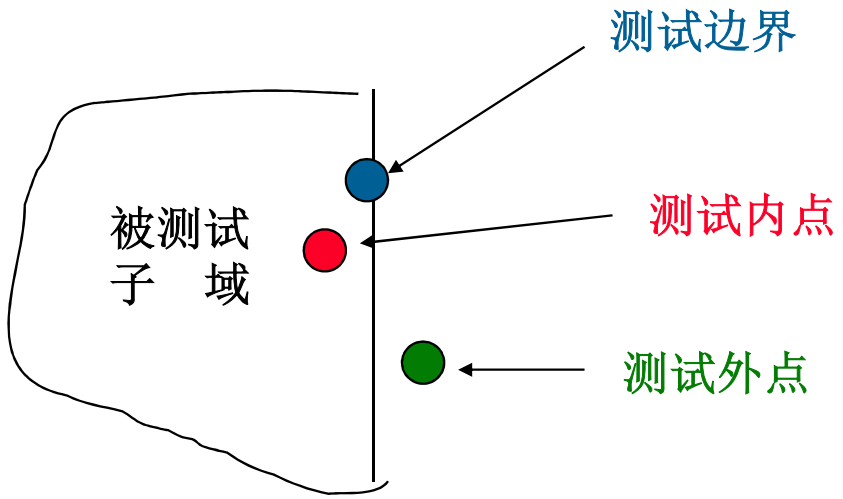
第二步：优化等价类（合并或拆分），并对每个等价类设计对应的测试用例。

练习：设有一个档案管理系统，要求用户输入以年月表示的日期。假设日期限定在1990年1月～2049年12月，并规定日期由6位数字字符组成，前4位表示年，后2位表示月。现用等价类划分法设计测试用例，对“日期检查”模块进行功能测试。

黑盒测试

黑盒测试——边界值分析

边界值分析法是对等价分类技术的补充，即在一个等价类中不是任选一个元素作为此等价类的代表进行测试，而是选择此等价类边界上的值。



黑盒测试

黑盒测试——边界值分析法

例：在做三角形计算时，要输入三角形的三个边长：A、B和C。 我们应注意到这三个数值应当满足

A＞0、B＞0、C＞0、

A＋B＞C、A＋C＞B、B＋C＞A，才能构成三角形。但如果把六个不等式中的任何一个大于号“＞”错写成大于等于号“≥”，那就不能构成三角形。问题恰出现在容易被疏忽的边界附近。

黑盒测试

边界值分析设计测试用例原则

(1) 如果输入数据给定了范围，则对于范围的边界，定义比边界值少1、边界值、比边界值多1的数值设计测试用例。例如[a, b]整数区域，则取a-1、a、a+1，b-1, b, b+1作为测试用例。

例1: 邮件收费规定 1～5 kg收费2元，则应根据数据精度要求，设计如下不同的测试用例:

0.9，1，1.1, 4.9, 5，5.1 kg

或 0.99，1，1.01, 4.99, 5，5.01 kg

(2) 如果规定了输入数据的个数N，则设计0个数据、1个数据、2个数据、N－1个数据、N个数据和N＋1个数据等的测试用例。

例2:一个输入文件可有1～255个记录则可分别设计有：0个、 1个、2个、254个、255个、256个记录的输入文件。

黑盒测试

(3) 如果没有指定值区域，则应取计算机所能表达的最大值和最小值作为测试用例。例如：整数、浮点数。

(4) 如果输入数据是有序集，则取该集中第一、第二，倒数第二和最后一个元素作为测试用例。

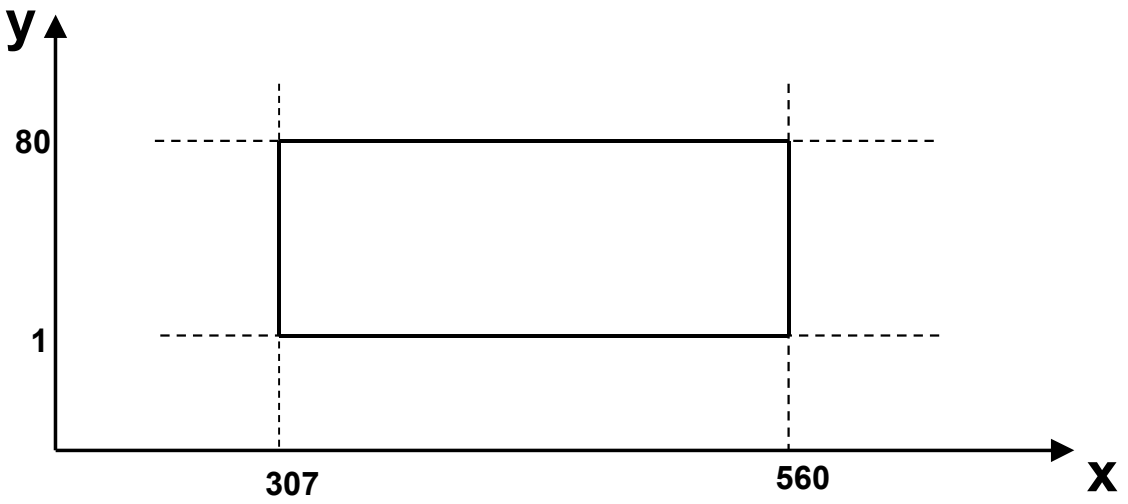
例：有两个输入变量 x(**307 ≤ x ≤ 560**) 和 y(**1 ≤ y ≤ 80**)的程序，请设计进行边界值分析测试的测试用例。

左边界：(306，50)、(307，50)、(308，50)

右边界：(559，50)、(560，50)、(561，50)

下边界：(400，0)、 (400，1)、(400，2)

上边界：(400，79)、(400，80)、(400，81)



黑盒测试

黑盒测试——错误推测法

黑盒测试

黑盒测试——等价类的组合

测试用例生成：被测模块通常有多个输入参数，如何对这些参数的等价类进行组合测试，来保证等价类的覆盖率，是测试用例设计需要进一步考虑的问题。

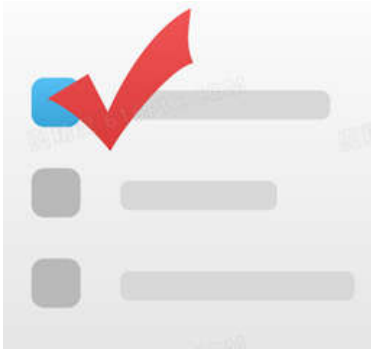
有效等价类的所有组合都集成到测试用例中，即覆盖所有有效等价类。任何一个组合都将设计成一个有效的测试用例，称为正面测试用例。

无效等价类只能与有效等价类进行组合，生成无效测试用例，称为负面测试用例。

等价类的组合将产生爆炸式的测试用例。如何有效地减少测试用例？

黑盒测试

黑盒测试——选取等价类组合的规则



- 由所有组合生成的测试用例按使用频率（或典型特征）设置优先级，这样仅对相关的测试用例（典型的组合用例）进行测试。
- 考虑包含边界值或者边界值组合的测试用例。
- 将一个等价类的每个代表值和其他等价类的每个代表值进行组合来设计测试用例（即双向组合代替完全组合）。
- 保证满足最小原则：一个等价类的每个划分至少取一个值并用于一个测试用例中。
- 无效等价类不与其他无效等价类进行组合。

黑盒测试

黑盒测试与白盒测试比较

	黑盒测试	白盒测试
优点	▶适用于各测试阶段 ▶从产品功能角度测试 ▶容易入手生成测试数据	▶可以构成测试数据使特定程序部分得到测试 ▶有一定的充分性度量手段 ▶可获得较多工具支持（反向工程）
缺点	▶某些代码段得不到测试 ▶如果规格说明有误则无法发现 ▶不易进行充分性度量	▶不易生成测试数据 ▶无法对未实现规格说明的部分测试 ▶工作量大，通常只用于单元测试，有引用局限

测试用例的有效性

测试用例的有效性评估

- 设计的测试用例，真能发现BUG吗？
- 为了覆盖而设计的测试用例，是真的能发现问题的测试用例吗？
- 测试用例设计得越来越多，如果删除部分，会不会发现不了问题？或者说，会不会没有减少问题的发现？

待测代码	if (age >= 14 && age <= 50)
	{
	Register(info);
	return true;
	}
	else
	return false;

测试用例的有效性

测试用例的有效性评估

待测代码	待测代码（注入变异）

if (age >= 14 && age <= 50)
{
Register(info);
return true;
}
else
return false;
	if (age >= 14 && age < 50)
	{
	Register(info);
	return true;
	}
	else
	return false;

age = 50，测试结果：false

测试用例	测试用例
age = 10，测试结果：false age = 30，测试结果：true age = 60，测试结果：false	age = 10，测试结果：false age = 30，测试结果：true age = 60，测试结果：false

测试用例的有效性

测试用例的有效性评估

待测代码
（注入变异）
	if (age >= 14 && age < 50)
	{
	Register(info);
	return true;
	}
	else
	return false;

age = 50，测试结果：false

测试用例	age = 10，测试结果：false age = 30，测试结果：true age = 60，测试结果：false

- 让注入变异后的业务代码作为“测试用例”，来测试“测试代码”。

- 测试用例的有效性说明，在其被测代码发生变化后（注入变异），这组测试用例中应至少有一个测试用例失败。
- 如果这组测试用例仍然全部通过测试，则说明这组测试用例可能存在测试有效性不足的问题。

测试用例的有效性

测试用例的有效性评估

- ✓ 为提高测试用例的有效性，总结出常用的注入变异规则。

运算符	
算术运算符	+/-，*/÷
关系运算符	>/<，==/!=,
逻辑运算符	||/&&
位运算符	&/|，>>/<<
赋值运算符	+=/-=，*=/÷=

语句	
条件语句	各类判断空语句
循环语句	1）条件恒为true / true 2）break / continue
异常处理语句	删除try...catch
赋值语句	=/==

数据结构	
初始值	1）bool型：true/false 2）指针为：空/非空
数值精度	整数 / 浮点数
格式	MM-DD-YYYY / DD-MM-YYYY

函数及类	
近似函数	GetSize() / GetLength()
重载函数	Operator(double, double) / Operator(int, int)
内存处理函数	1）删除内存指针判空 2）删除回收内存语句
赋值语句	=/==

测试用例的有效性

测试用例的有效性评估

✓ 提高测试用例的有效性，还有很多方法等待同学们去发现和总结：

- 代码注入：向代码注入变异，验证测试用例能否发现注入的BUG。
- 接口注入：修改接口（特别是类的继承中）的返回，看测试用例能否发现接口的BUG。
- ……

提高测试的有效性，使得测试用例变得更高效地发现问题，并能让无效用例可被识别、清理。