软件工程导论 -

第7章 实现-软件测试

主 讲 人：刘文洁
工作单位：西北工业大学计算机学院
Email ：liuwenjie@nwpu.edu.cn

第7章　软件测试

- 7.1 检验的基本概念
- 7.2 测试的基本概念
- 7.3 白盒法
- 7.4 黑盒法
- 7.5 测试步骤
- 7.6 单元测试
- 7.7 集成测试
- 7.8 确认测试
- 7.9 调试
- 7.10 软件可靠性

7.1 检验的基本概念

软件检验的手段：

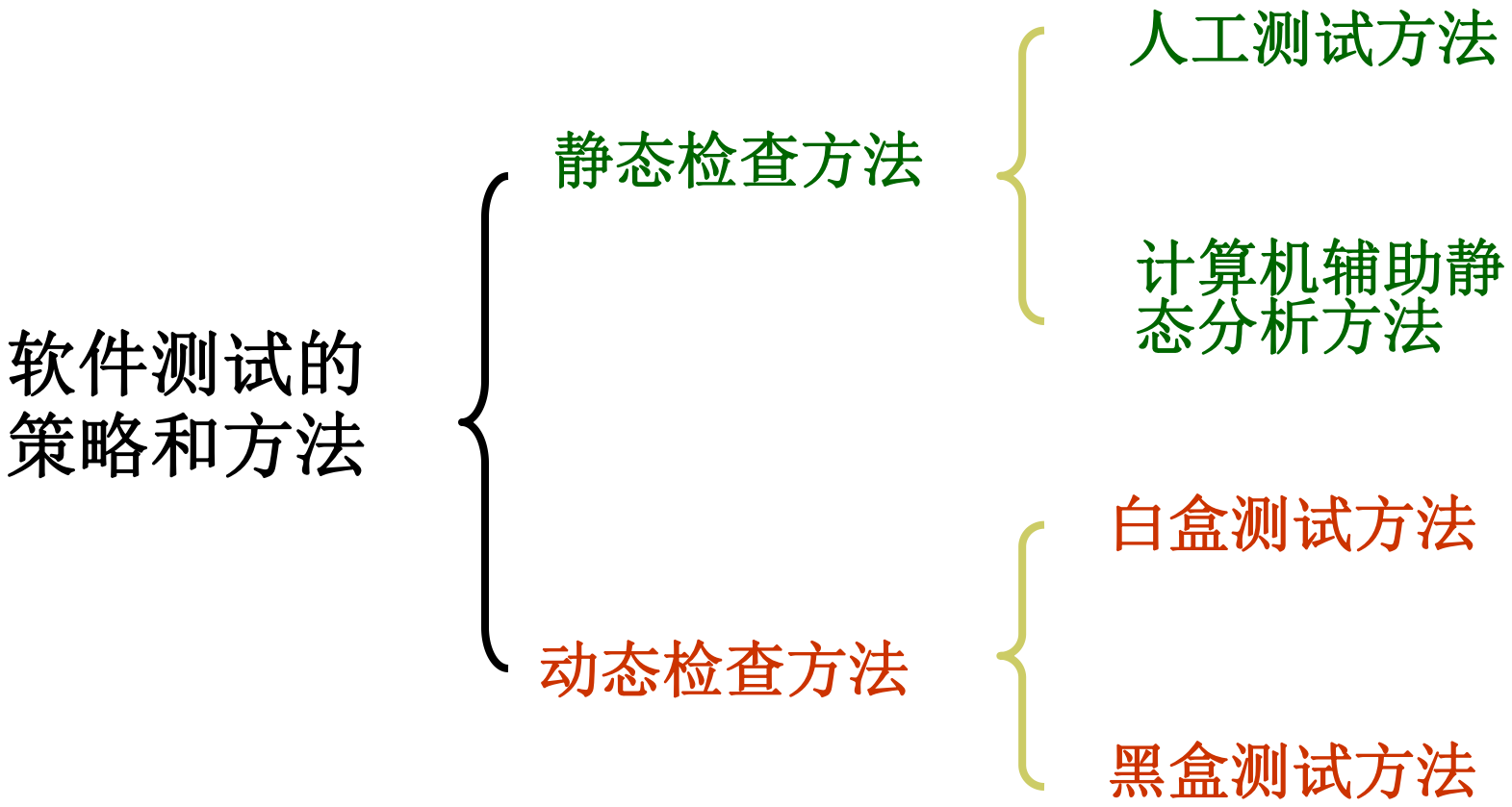
- 动态检查 — 测试
- 静态检查 — 人工评审软件文档或程序
- 正确性证明

检查时只以找出错误为目的。

发现错误之后，“排错”应由软件开发方完成。

最终的检验验收工作应由检验部门来实施。

软件检验与测试方法


软件测试的
策略和方法
静态检查方法
人工测试方法
计算机辅助静
态分析方法
动态检查方法
白盒测试方法
黑盒测试方法

静态检查（评审、review）：

- 基本特征是在对软件进行分析、检查和审阅，不实际运行被测试的软件。
  - 静态测试约可找出**30～70%**的逻辑设计错误.
  - 对需求规格说明书、软件设计说明书、源程序做检查和审阅
    - 包括：
      - 是否符合标准和规范；
      - 通过结构分析、流图分析、符号执行指出软件缺陷。

软件评审

需求设计 -- 需求设计复查
概要设计 -- 概要设计复查
详细设计 -- 详细设计复查
程序 -- 程序复查走查

评审针对软件，而不是作者

评审发现错误而不是纠正错误

评审会不要持续过长

动态检查（测试）：

- 通过运行软件来检验软件的动态行为和运行结果的正确性。

- 动态检查的两个基本要素：
  - 被测试程序
  - 测试数据（测试用例）

- 动态检查方法：
  - (1) 选取定义域有效值,或定义域外无效值；
  - (2) 对已选取值决定*预期的结果*；
  - (3) 用选取值执行程序；
  - (4) *执行结果*与*预期的结果*相比,不吻和程序有错。

软件开发活动模型- 静态检查与动态测试

需求阶段	设计阶段	实现阶段	测试阶段	验收阶段
• 用例情景测试 • 原型走查 • 模型走查 • 需求评审	• 模型走查 • 原型走查 • 设计评审	• 代码走查 • 接口分析 • 文档评估		
• 制定测试计划	• 制定测试计划 • 测试设计	• 编写测试用例 • 制定测试过程 • 单元测试	• 制定测试过程 • 集成测试 • 系统测试	• α 测试 • β 测试 • 验收测试
回归测试，质量保证				

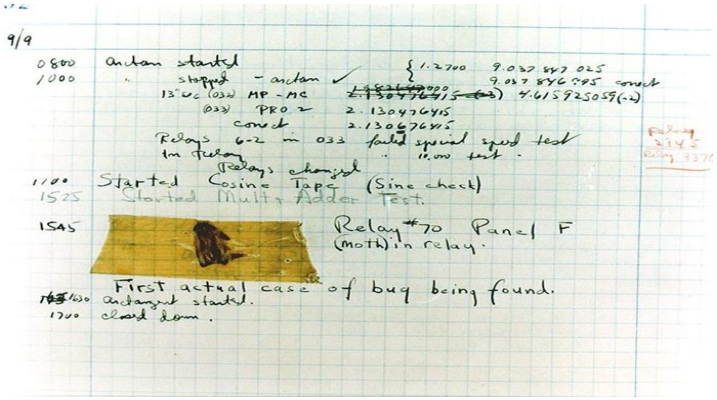
7.2 测试的基本概念

软件测试的问题

- 软件缺陷是什么？
- 谁执行测试？
  - 开发者？
  - 单独的测试人员？
  - 两方面人员？
- 测试什么？
  - 每个部分都测试？
  - 测试软件中高风险部分？
- 什么时候测试？
- 怎样测试？
- 测试应进行到什么程度？

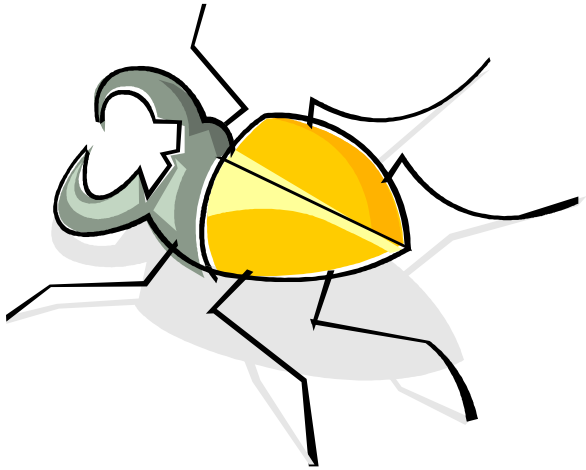
软件错误无处不在

- 只要是人编写的软件，就不能避免软件错误的发生
  - 软件的缺陷通常被称为Bug。
  - 1946年，格蕾丝·赫柏Grace Hopper，加入了哈佛大学的计算实验室，1946年，她在发生故障的Mark Ⅱ计算机的继电器触点里，找到了一只被夹扁的小飞蛾，正是这只小虫子“卡”住了机器的运行。赫柏顺手将飞蛾夹在工作笔记里，并诙谐地把程序故障称为“bug”。后来演变成计算机行业的专业术语。虽然现代电脑再也不可能夹扁任何飞蛾，大家还是习惯地把排除程序故障叫做Debug。



软件缺陷是什么 --- 描述软件失败的术语

- 缺点 (defect)
- 谬误 (fault)
- 问题 (problem)
- 错误 (error)
- 异常 (anomaly)
- 偏差 (variance)
- 失败 (failure)



- 缺陷 (bug)

从软件错误到软件失效

- 软件问题产生的过程：软件错误（error）->软件缺陷（ defect ）->软件故障（ fault ）->软件失效（ failure ）
  - 软件错误是一种人为错误。
  - 一个软件错误必定产生一个或多个软件缺陷。
  - 当一个软件缺陷被激活时，便产生一个软件故障；同一个软件缺陷在不同条件下被激活，可能产生不同的软件故障。
  - 软件故障如果没有及时的容错措施加以处理，便不可避免地导致软件失效；同一个软件故障在不同条件下可能产生不同的软件失效。

软件缺陷的例子

- 程序异常终止；
- 未达到需求说明书指明的要求；
- 出现了需求说明书中指明不会出现的错误；
- 功能超出了需求说明书指明的范围；
- 未达到需求说明书未指明但是应达到的要求；
- 难以理解，不易使用，运行速度缓慢；

软件测试的现状

- 与一些发达国家相比，国内测试工作还存在一定的差距。国内测试人员所占比例小。
- 微软的开发工程师与测试工程师的比例是1 ： 2,国内一般公司是6 ： 1。
- 与发达国家相比，我们的差距主要在测试意识，测试理论的研究，测试工具软件的开发以及从业人员的数量等方面。
- 优秀测试人员匮乏。
- 人工功能测试人员需求较大。

7.2.1 测试的基本概念以及目标

- 测试的目标：通过程序的执行，发现在设计和制造阶段植入的**bug.**
- 测试 = 发现问题

≠ 证明程序的正确性

测试的目标 = 具有现实意义的测试

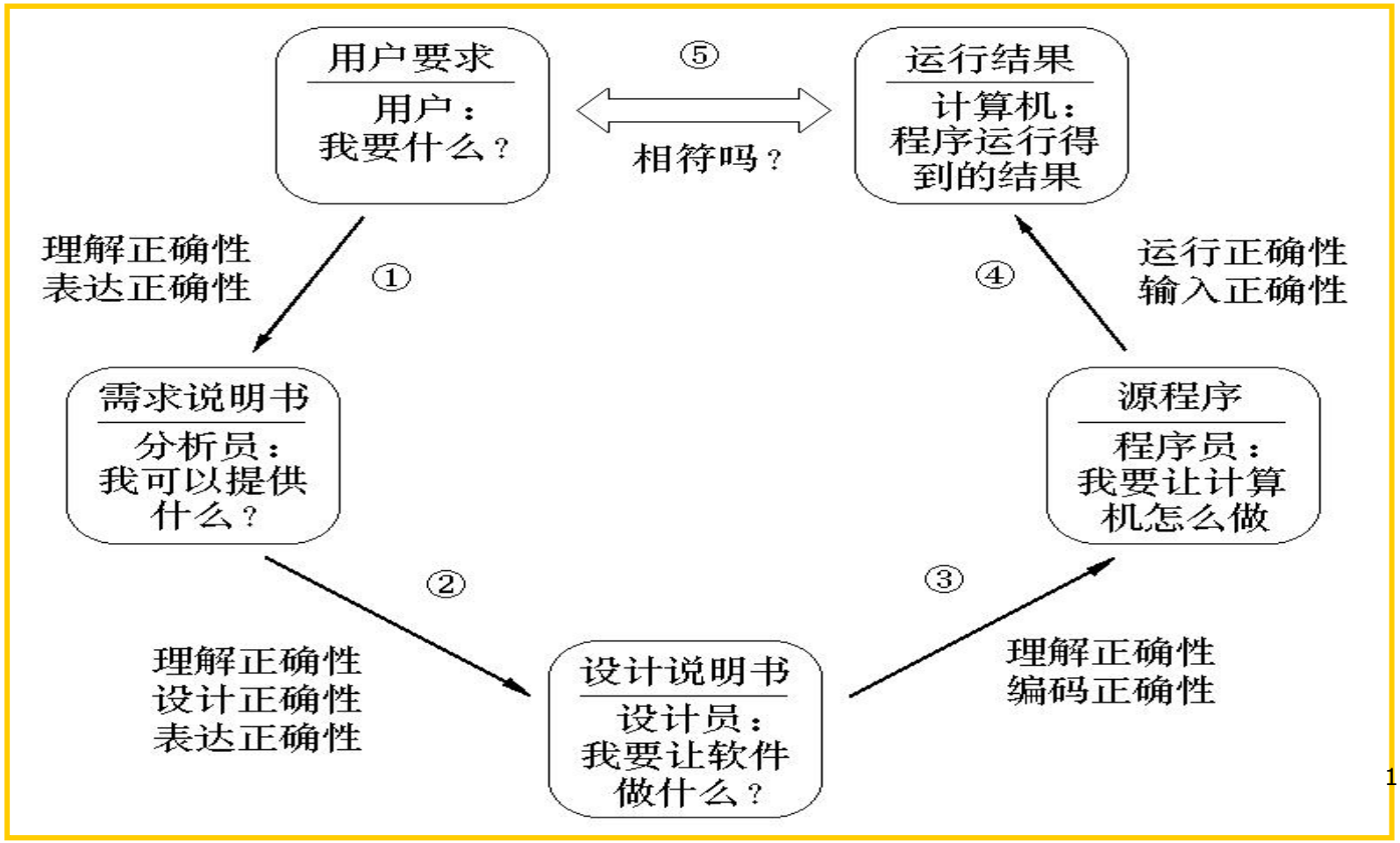
设计高效率的测试项

✓一个好的测试用例在于它发现了至今未发现的错误。

✓ 一个成功的测试是发现了至今未发现的错误的测试。

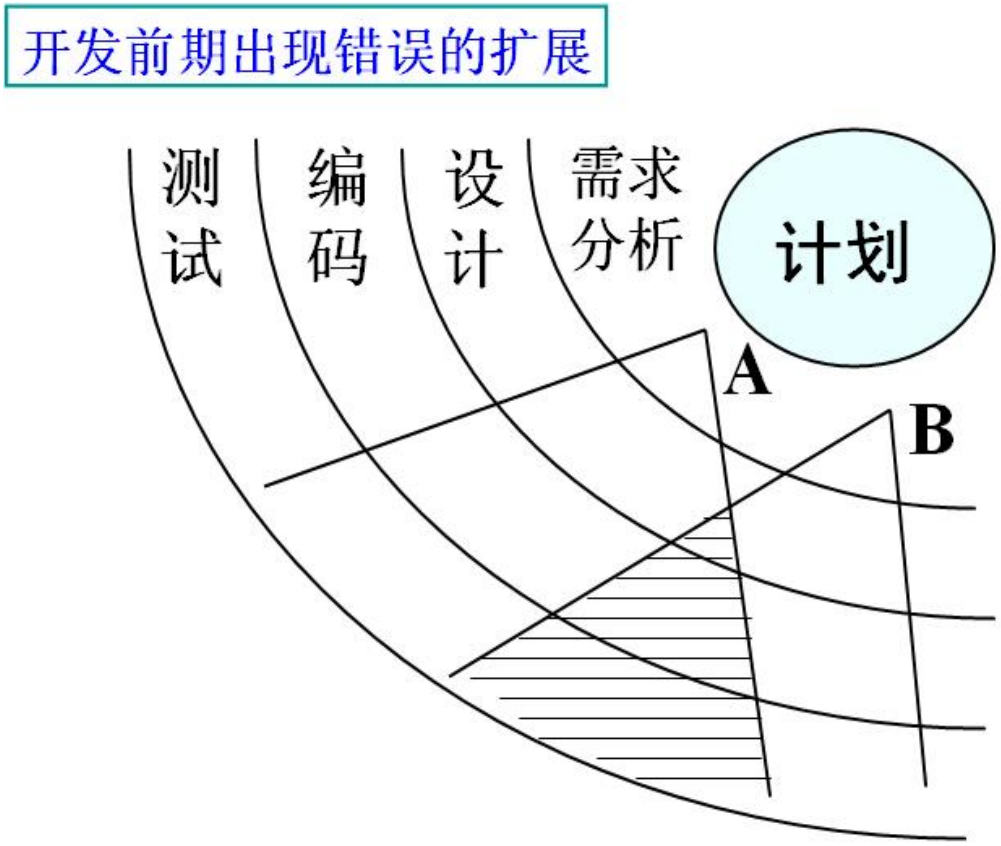
7.2.2 软件测试准则(1)

- 所有测试都应该能追溯到用户需求。



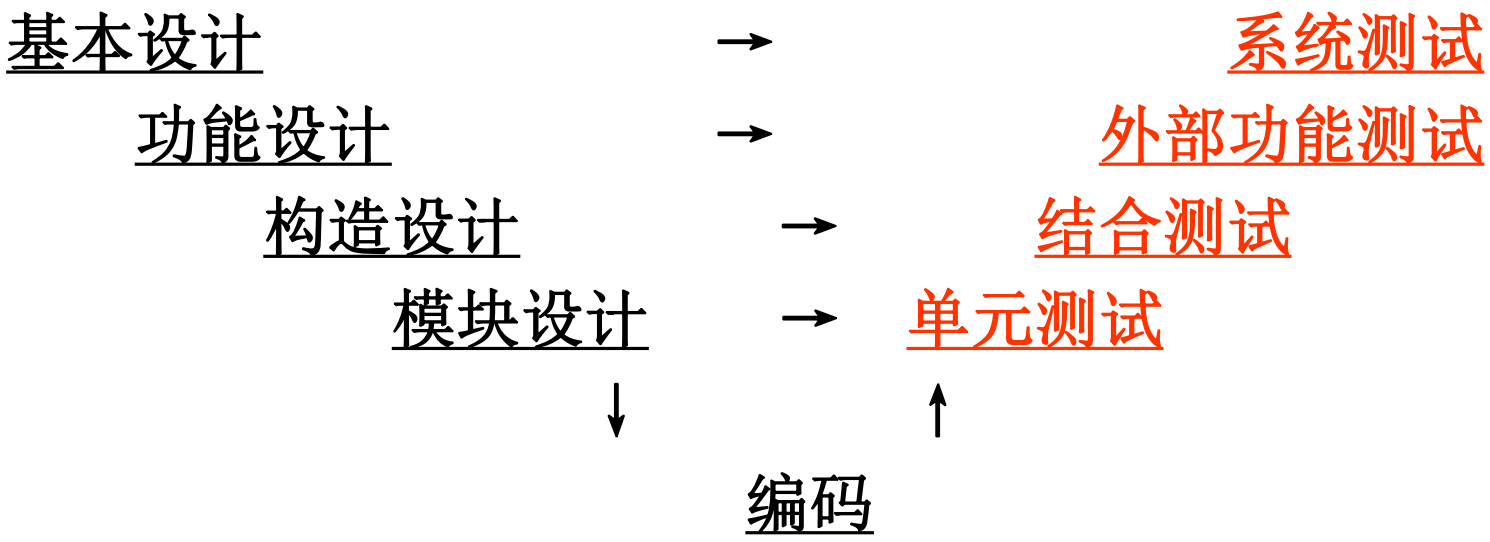
软件测试准则(2)

- 应当把“尽早地和不断地进行软件测试”作为软件开发者的座右铭。
  （软件测试不等于程序测试。软件测试应贯穿于软件定义与开发的整个期间；）



开发工程和测试的对应关系

V字型的测试工程



软件测试准则(3)

- pareto原则：测试发现的错误中的80%很可能是由程序中20%的模块造成的。
- 应该从“小规模”测试开始，并逐步进行“大规模”测试。
- 测试用例应由输入数据和预期的输出结果两部分组成，并兼顾合理的输入和不合理的输入数据。
- 穷举测试是不可能的。
- 为了达到最佳的测试效果，应该由独立的第三方从事测试工作。
- 程序修改后要回归测试。
- 应长期保留测试用例，直至系统废弃。

测试项的设计

测试项目 = 测试输入数据 + 预期的输出结果

程序的输入是什么？有什么功能什么处理？

对于给的输入，应该有什么样的输出？

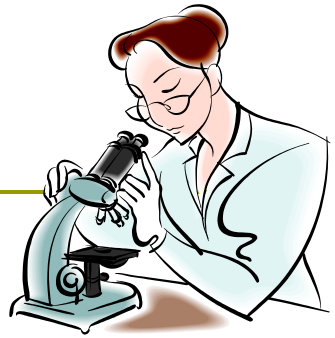
- 测试项目不应重复
- 测试项目，测试观点不应遗漏；除了检查应作的，还要检查是否作了不应作的
- 结果的确认方法应妥当（debug，日志，目视等）
- 测试项目应该进行复查
- 测试用例应该长期保留

测试结果要真实、可靠，
测试人员要对测试结果负责！

在近乎无限的测试项目中，抽出可以有效发现bug的测试项。

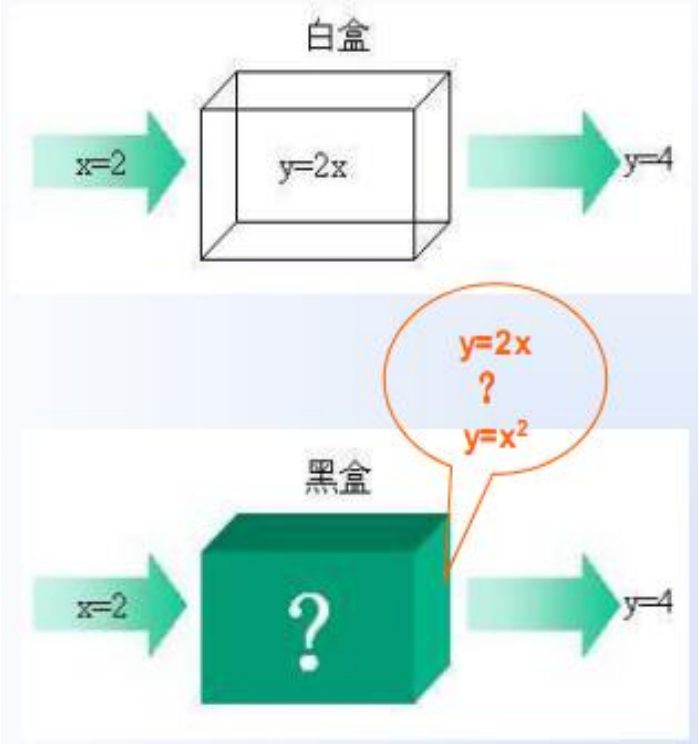
为了达到这个目的，需要使用各种测试设计技法。

7.3 白盒法

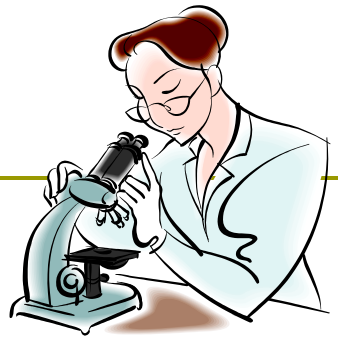


也叫玻璃盒测试(Glass Box Testing)

对软件的过程性细节做细致的检查。这一方法是把测试对象看作一个打开的盒子，它允许测试人员利用程序内部的逻辑结构及有关信息，来设计或选择测试用例，对程序所有逻辑路径进行测试。



白盒测试的方法



- 逻辑覆盖
  - 语句覆盖
  - 判定覆盖
  - 条件覆盖
  - 判定-条件覆盖
  - 条件组合

面向单语句的测试准则

- 路径覆盖 → 面向语句间控制的测试

路径测试:从流程图上讲，程序的一次执行对应于从入口到出口的一条路径，针对路径的测试即为路径测试。

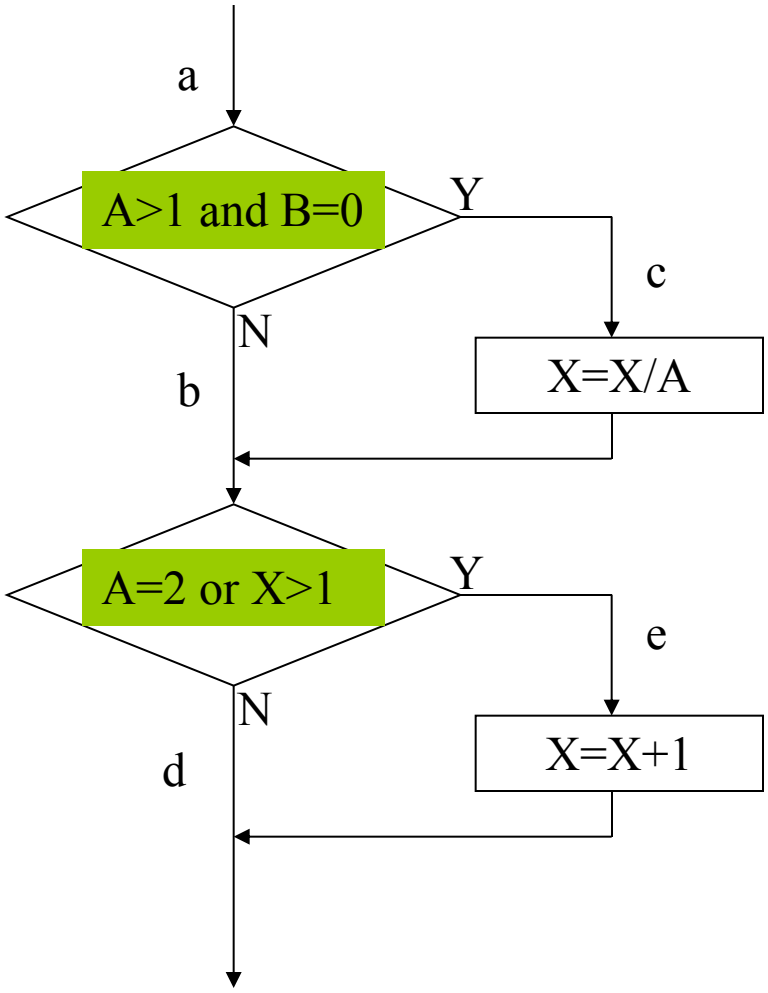
为什么要白盒测试

- 逻辑错误往往发生在不太经过的程序路径。
- 打字错误是随机的。
- *注意：白盒测试做多做少与产品形态有关

白盒法

- 模块中所有独立路径至少被使用一次；
- 所有逻辑值均须测试真（**TRUE**）和假（**FALSE**）；
- 在上下边界及可以操作的范围内运行所有循环；

例题



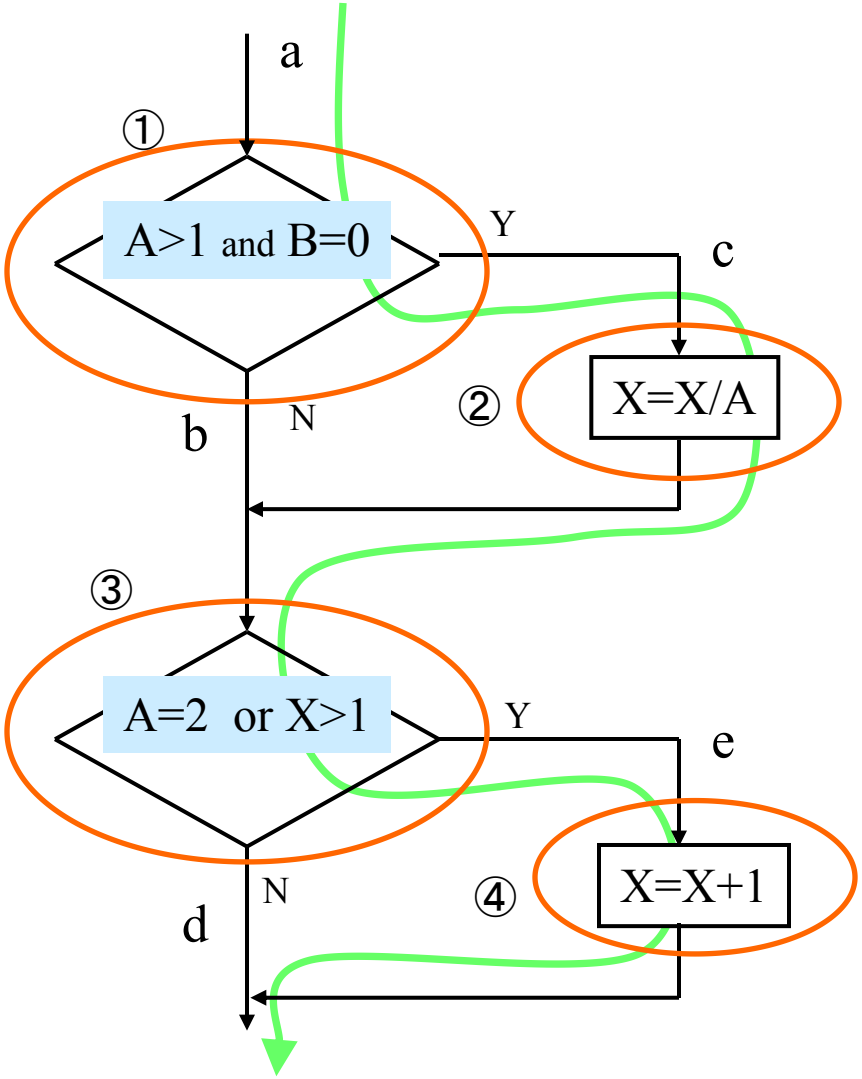

```
int MySample(int a, b, x)
{

   if ( (a>1) && (b==0) )
   {
        x = x/a;
    }

   if ( (a==2) || (x>1) )
   {
        x = x+1;
    }

    return x;
}
```

7.3.1 语句覆盖法–每个语句至少被执行一遍



例题由4条语句组成

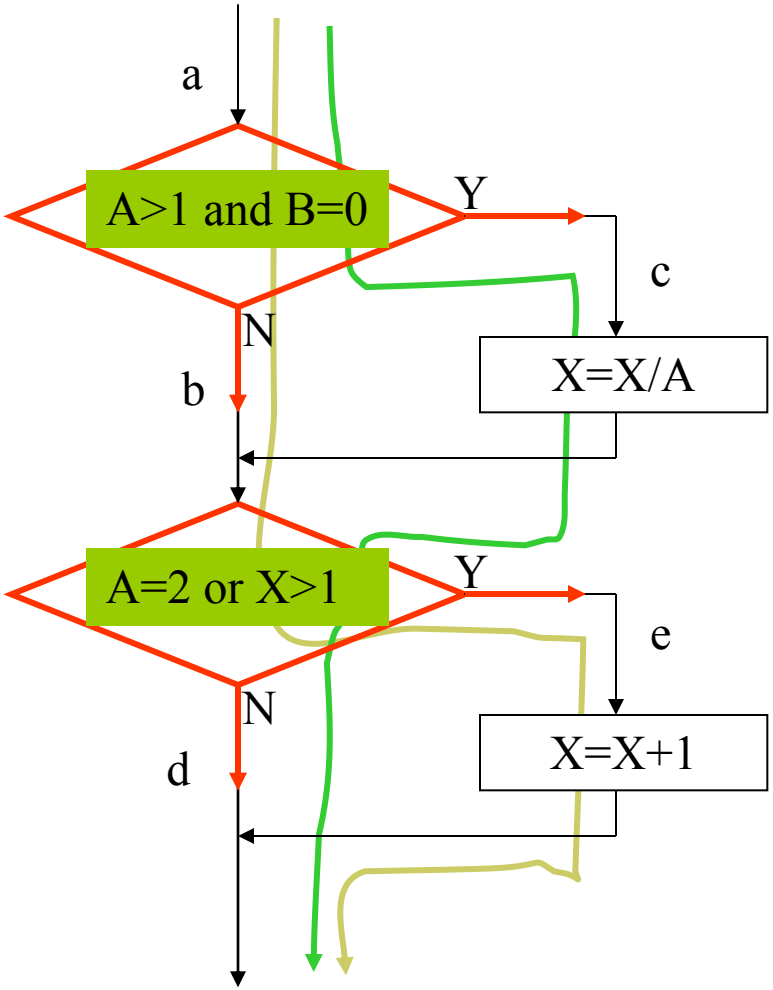
①③判定条件 ②④赋值语句

覆盖全部语句的通路是ace

可选择的输入数据例为：

A=2，B=0，X=3

7.3.2 判定覆盖法–每一个分支至少被执行一遍



覆盖

①③判定条件

的通路

例**acd, abe**

可选择的输入数据例为：

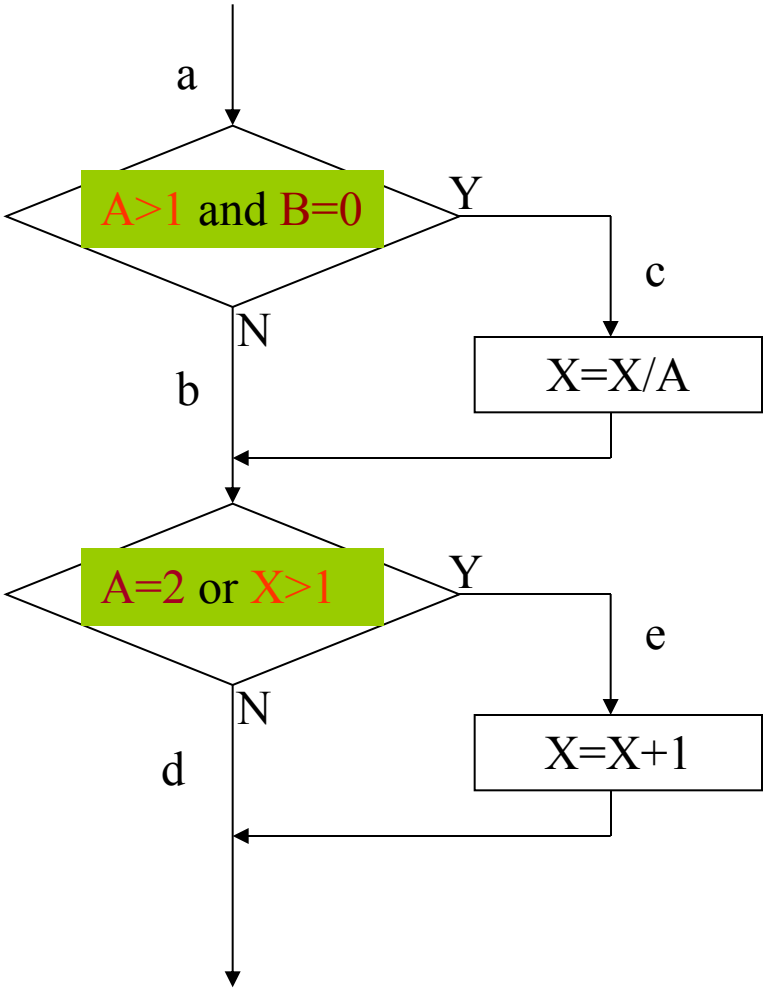
A=3，B=0，X=1 (acd)

T && T => T
F || F => F

A=2，B=1，X=3 (abe)

T && F => F
T || T => T

7.3.3 条件覆盖法–每个条件获得各种可能的结果



共有四个条件

A>1，B=0，A=2，X>1

覆盖四个条件真假值
的可选择的输入数据例为：

A=2，B=0，X=4 (ace)

```
T &&  T
T ||   T
```

A=1，B=1，X=1 (abd)

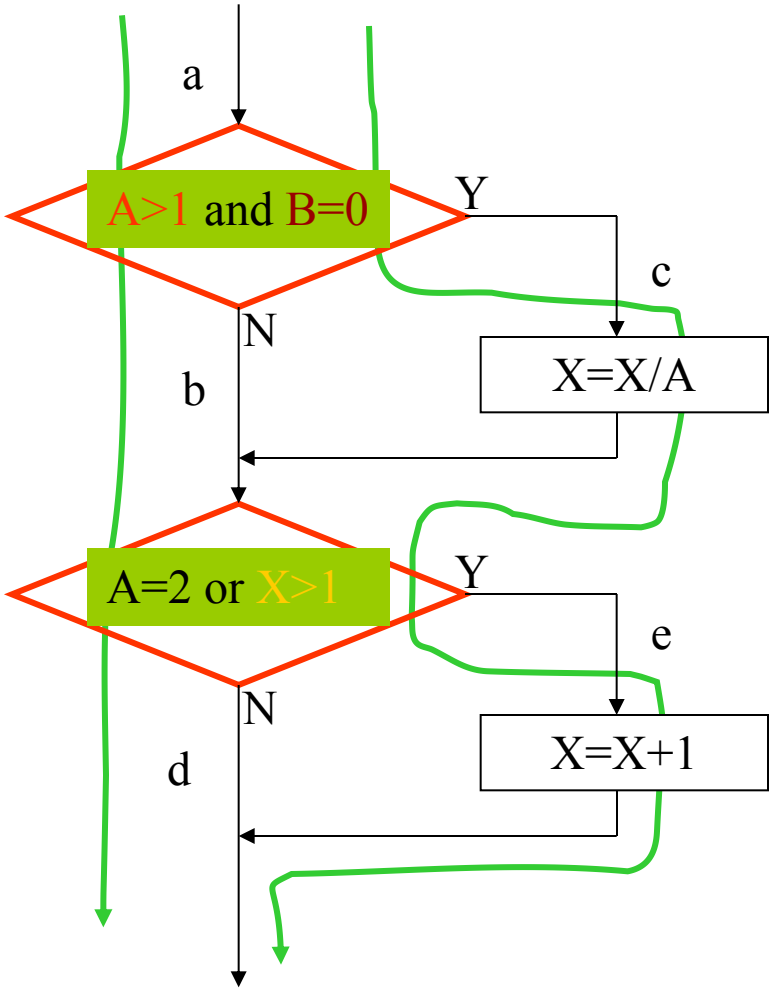
```
F &&  F
F ||   F
```

补充说明

- “条件覆盖”通常比“判定覆盖”强，因为它使一个判定中的每一个条件都取到了两个不同的结果，而判定覆盖则不保证这一点。

- “条件覆盖”并不包含“判定覆盖”。
  - 例如：语句: if (A and B) then S
  - 设计测试用例使其满足"条件覆盖":
  - test case 1: A:true, B:false,
  - test case 2: A:false, B:true,
  - 以上2个测试用例都未能使语句S得以执行（true分支没有得到覆盖）。

7.3.4 判定/条件覆盖法—每一个分支，每个条件至少被执行一遍



覆盖两个判定分枝，
四个条件真假值的可选择的输入数据例为：

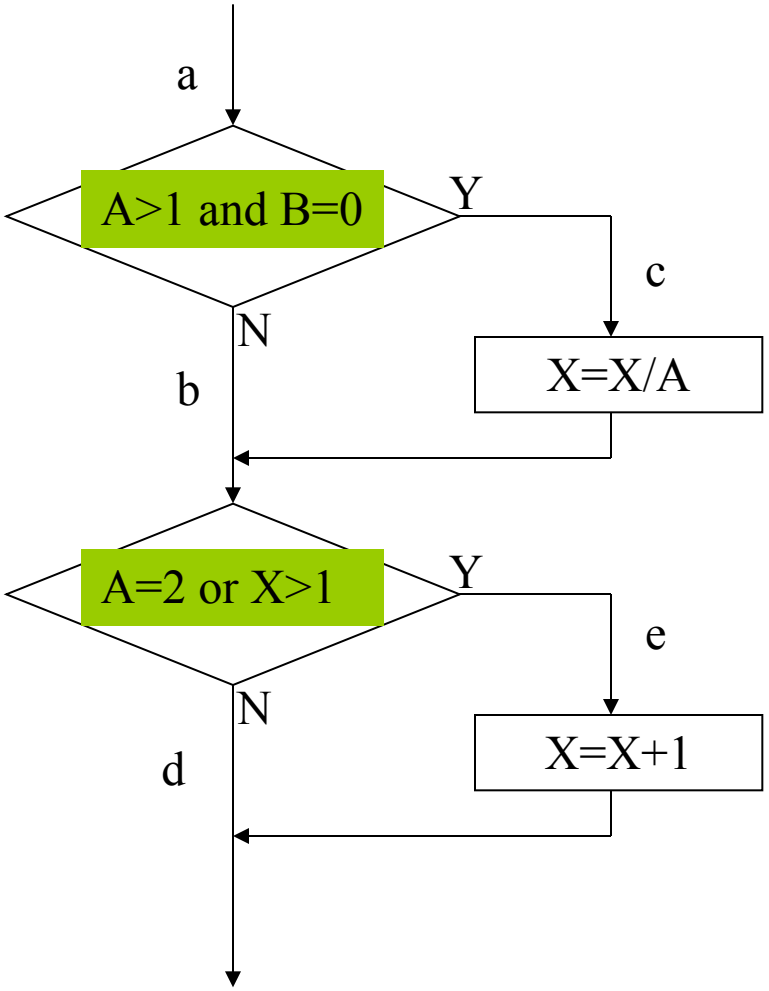
A=2，B=0，X=4 (ace)

T && T => T
T || T => T

A=1，B=1，X=1 (abd)

F && F => F
F || F => F

7.3.5 条件组合覆盖法–每个条件的可能组合获得各种可能的结果



上面判断的组合

①真与真 → A>1, B=0

②真与假 → A>1, B<>0

③假与真 → A<=1, B=0

④假与假 → A<=1, B <> 0

下面判断的组合

⑤真与真 → A=2, X>1

⑥真与假 → A=2, X<=1

⑦假与真 → A<>2, X>1

⑧假与假 → A<>2, X <= 1

条件组合的分析

第2个条件的组合 / 第1个条件的组合	⑤A＝2、X＞1	⑥A＝2、X≦1	⑦A≠2、X＞1	⑧A≠2、X≦1
①A＞1、B＝0	○	○	○	○
②A＞1、B≠0	○	○	○	○
③A≦1、B＝0	×	×	○	○
④A≦1、B≠0	×	×	○	○

○:条件可以同时成立的组合

×:条件不可以同时成立的组合

测试项的设计

- （①A＞1、B＝0）且（⑤A＝2 、X＞1）
  上面条件成立的例
  a的地方输入：A＝2、B＝0、X＝4

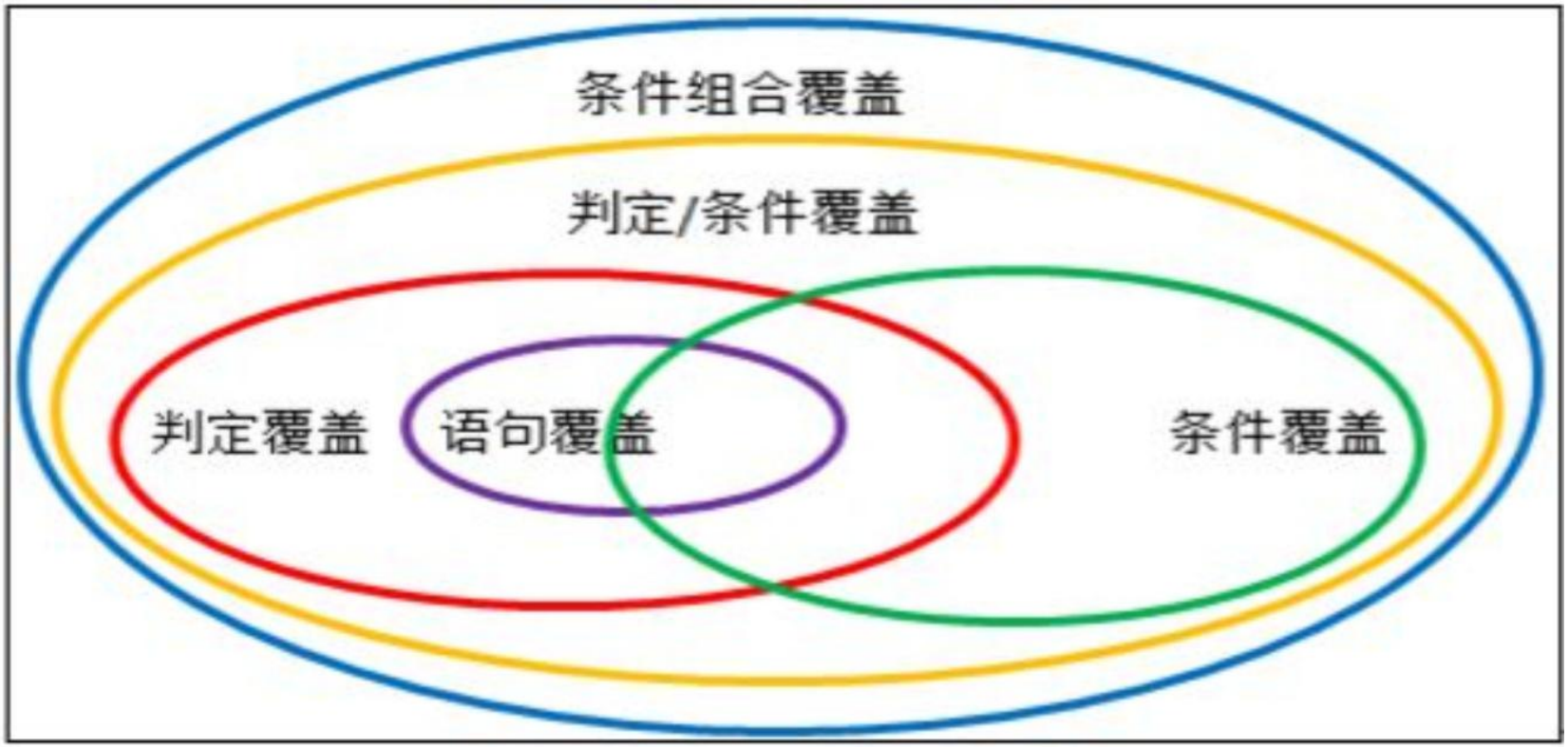
- （②A＞1、B≠0）且（⑥A＝2 、X≦1）
  上面条件成立的例
  a的地方输入：A＝2、B＝1、X＝1

- （③A≦1、B＝0）且（⑦A≠2、X＞1）
  上面条件成立的例
  a的地方输入：A＝1、B＝0、X＝2

- （④A≦1、B≠0）且（⑧A≠2、X≦1）
  上面条件成立的例
  a的地方输入：A＝1、B＝1、X＝1

五种覆盖的关系

- 条件组合覆盖，是最强的代码覆盖方法。



基本路径测试

- 一条路径是基本路径如果：
  - 是一条从起始节点到终止节点的路径.
  - 至少包含一条其它基本路径没有包含的边.

(至少引入一个新处理语句或一个新判断的程序通路.)

注意：对于循环而言，基本路径应包含不执行循环和执行一次循环体。

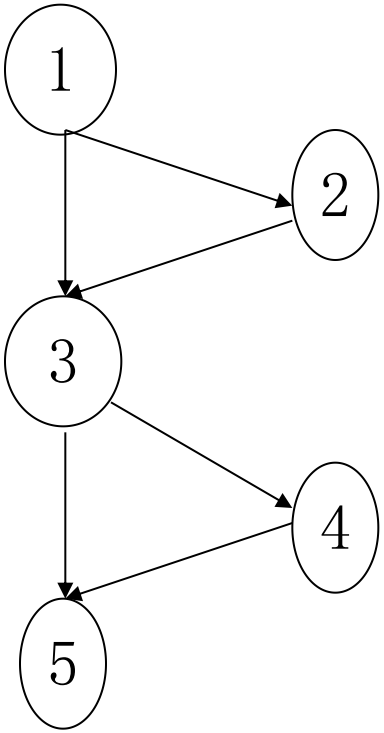
- 程序的环形复杂度决定了程序中独立路径的数量。

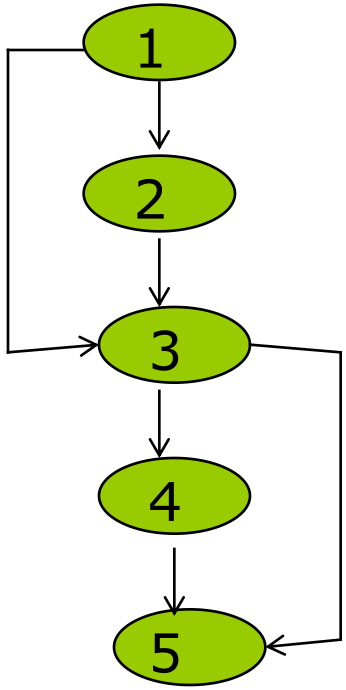
基本路径测试实施步骤：

1. 画出程序的控制流图;
2. 计算流图G的环路复杂性V(G);
3. 确定只包含独立路径的基本路径集;
4. 根据上面的独立路径，设计测试用例，使程序分别沿上面的独立路径执行。

基本路径举例1

- **左图实例的**控制流（程序）图对应的独立路径有3条。





12345

1235

135

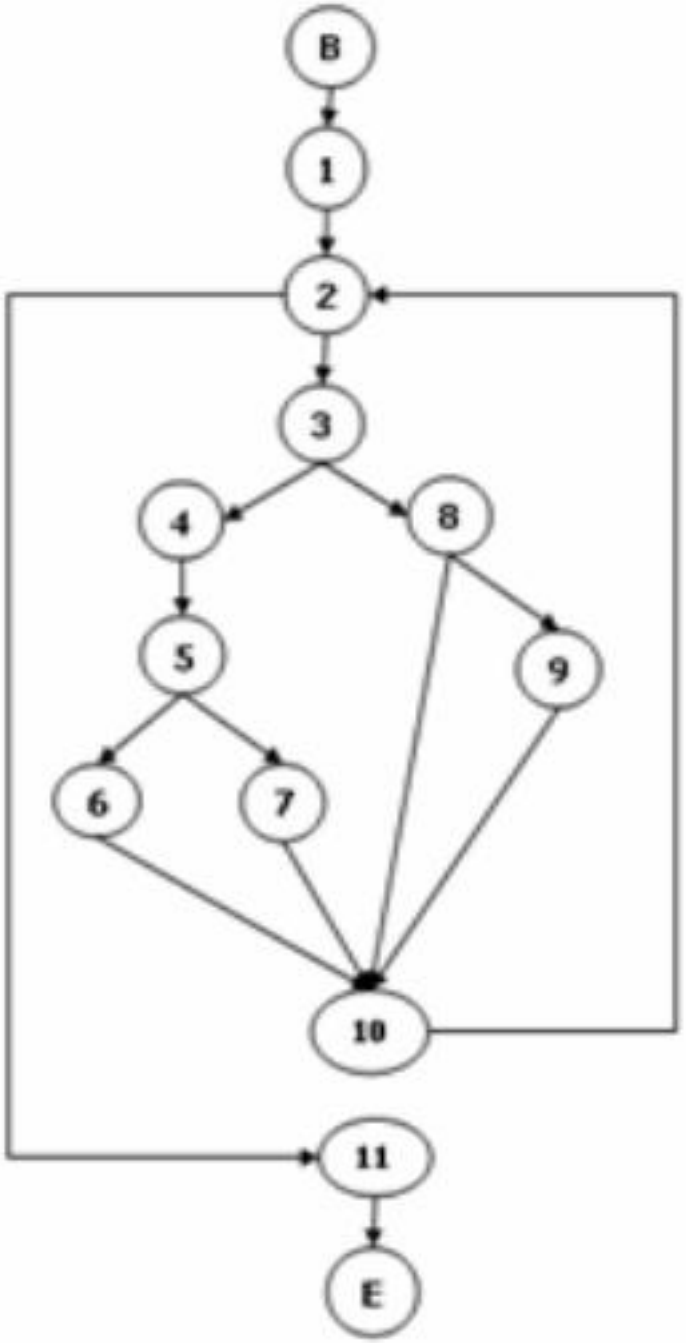
1345

12345

1235

基本路径举例2

1. 1-2-11
2. 1-2-3-4-5-6-10-2-11
3. 1-2-3-4-5-7-10-2-11
4. 1-2-3-8-9-10-2-11
5. 1-2-3-8-10-2-11



循环测试

- 整个跳过循环
- 只有一次通过循环
- 两次通过循环
- 循环次数范围内
- 循环次数-1次和+1次

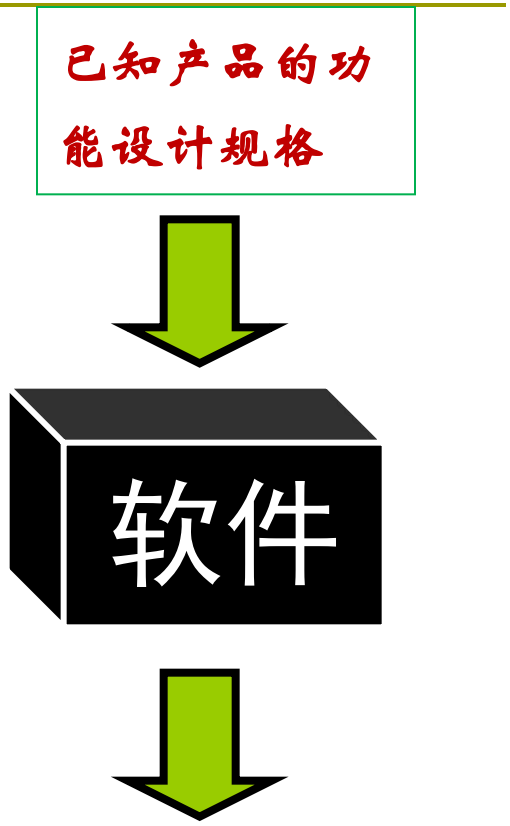
- 补充：循环最简单的测试方法，是把循环当分支结构，只测试进入循环和不进入循环体两种情况，这种简化循环意义下的路径覆盖成为“Z路径覆盖”。

白盒测试的设计步骤/方法

- ①明确程序的构造，流程
- ②确认所有应该网罗的对象（命令，条件分支）
- ③追加使得程序主要处理/通路获得执行的对象
- ④设计输入数据
- ⑤设计预测值

7.4 黑盒法

- 等价分类法
  程序的输入集合分割为有限的等价类
- 边缘值法
  等价类中的代表值取值时，采用等价类的边界值
- 错误推测法
  输入可能会发生错误的值
- 因果图法
  测试项目的组合
- 状态迁移法
  功能中含有状态迁移的情况使用的测试法



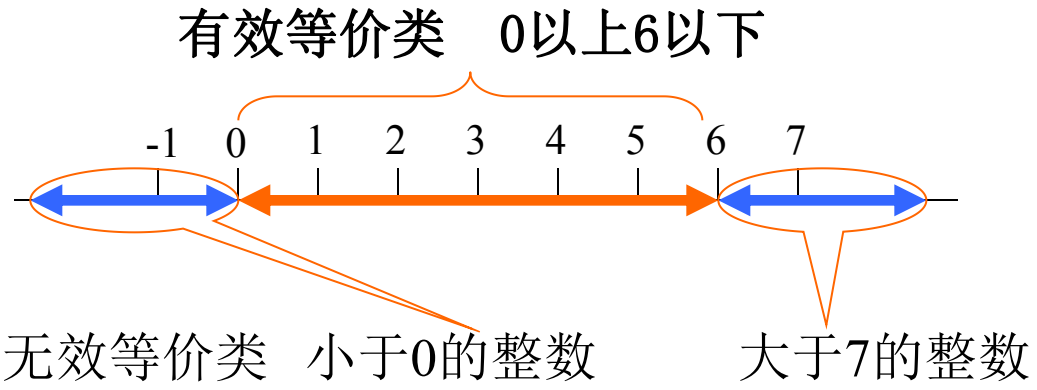
等价类划分基本思想

- 为什么需要等价类划分
  - 穷举测试不可能；
  - 希望达到测试的完备和无冗余；
- 什么是等价类划分
  - 等价类是指某个输入域的子集合，根据等价关系，将输入数据集合划分为互不相交的子集（无冗余性）；
  - 这些子集的并是整个集合（完备性）；
  - 在该子集合中，各个输入数据对于揭露程序中的错误都是等效的；
  - 测试某等价类的代表值就等于对这一类其它值的测试；

7.4.1 等价分类法—使用具有代表性的测试项(1)

- 程序的输入集合分割为有限的等价类，选取代表值进行测试的方法。
  具体来说，最基本可以将程序的输入分割为有效值（有效等价类）和无效值（无效等价）。

例：输入参数为0以上6以下的整数



7.4.1 等价分类法—使用具有代表性的测试项(2)

- 例题

8文字（英数字）以内过程的文件名为正常文件名

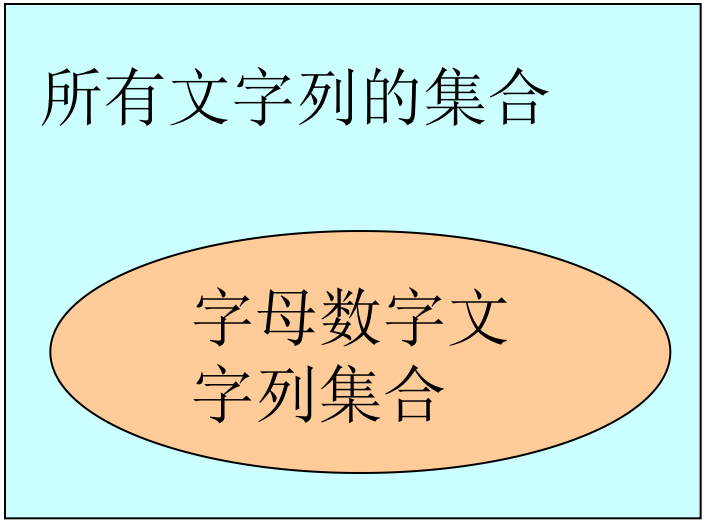
8文字（英数字）的组合有约3兆个 →

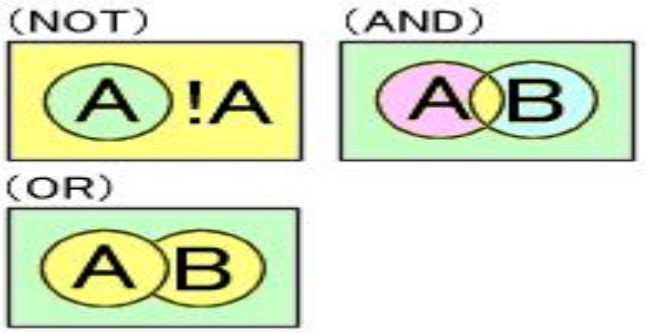
使用测试项目设计技法 → 等价分类法

7.4.1 等价分类法—使用具有代表性的测试项(3)

- 分割为不重合的等价类集合

例分为两个集合：英数字的集合和以外的集合

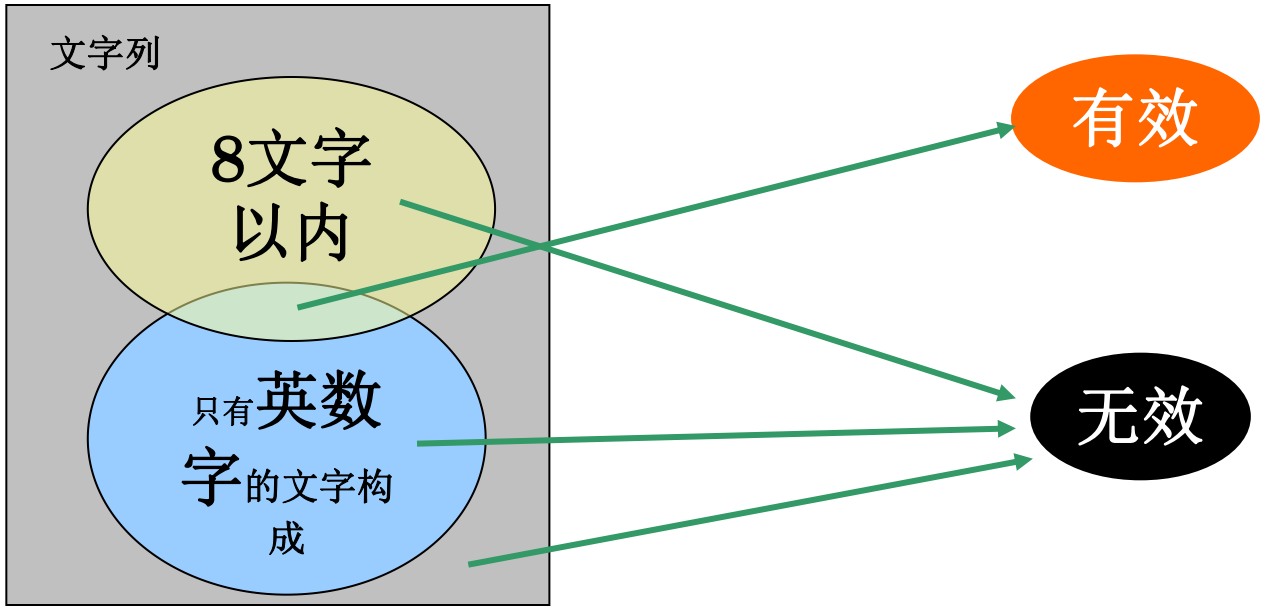




等价类的表示法

7.4.1 等价分类法—使用具有代表性的测试项(4)

- 通过对输入和输出进行分析，可进一步细化等价分类。



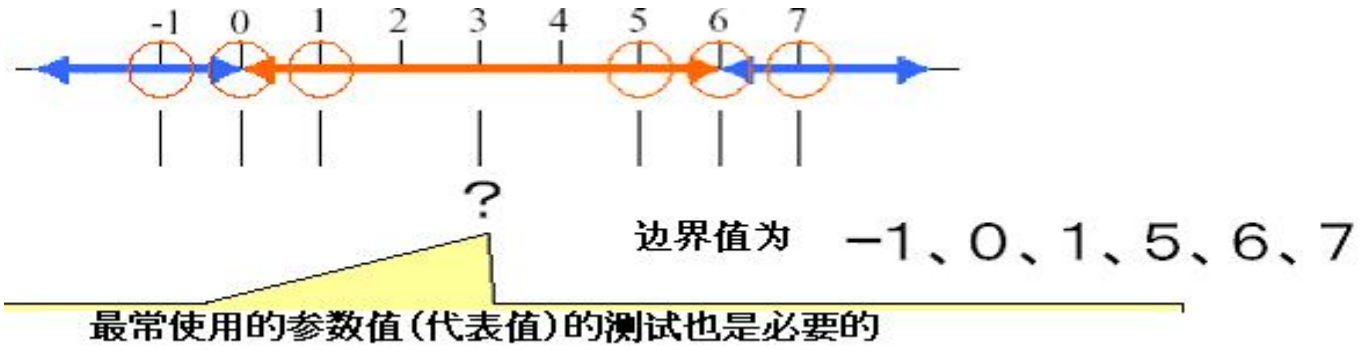
7.4.1 等价分类法—使用具有代表性的测试项(5)

- 测试项目表

番号	输入	等价类	输出	确认
1	1文字以上8文字以内的英数字	正确的文件名	有效	
2	1文字以上8文字以内的其他文字	错误的文件名	无效	
3	9文字以上的英数字	错误的文件名	无效	
4	其他（9文字以上的其他文字）	错误的文件名	无效	

7.4.2 边缘值分类法–等价类的边界值(1)

- 注意等价类的边界，对边界进行测试
- 程序的边界处理容易出现错误的地方
  - if (x < 8) 与 if (x ≦ 8)
  - while() 与 until()
  - for {i = 0; i < 8; i++ } 与 for {i = 1; i < 8; i++ }
- 首先确定等价类，然后决定边界值



7.4.2 边缘值分类法–等价类的边界值(2)

- 根据等价类进行边界值设计的例

8文字以内英数字文件名的检查功能

以文件名的文字数为中心分析边界值+代表值边界值也包含**0**，**9**，**A, z**

Bypass	文件名**0**文字	**Null**	无效
Once	文件名**1**文字	**"0" or "9" or "a" or "Z"**	有效
Twice	文件名**2**文字	**"9a" or "0Z"**	有效
Typical	文件名**5**文字	**"by18"**	有效
Max	文件名**8**文字	**"Za0A9z5C"**	有效
Max +1	文件名**9**文字	**"Za0A9z5Ca"**	无效
Max -1	文件名**7**文字	**"Za0A9z5"**	有效
Min	**1**文字（同**Once** ）		有效
Min +1	**2**文字（同**Twice** ）		有效
Max -1	**0**文字（同**Bypass** ）		无效
Null	**0**文字（同**Bypass** ）		无效
Negative	负个数的文件名		不存在
1文字以上**8**文字以内的其他文字		**"/" or ":ab" or "@1"**	无效
9文字以上的其他文字		**"+abc*32yxs"**	无效

总结

- 边界值分析与等价类划分的不同：

(1)边界值分析不是从某等价类中随便挑一个作为代表，而是使这个等价类的每个边界都要作为测试条件。

(2)边界值分析不仅考虑输入条件，还要考虑输出空间产生的测试情况。

软件边界类似于悬崖

7.4.3 错误推测法

- 根据过去的开发经验，程序的构造，处理等预测可能发生的bug。
- 等价类法和边缘值法的一个补充

接口错误 功能不足 错误对应 边界值对应的错误 计算的错误	初期状态和启动后的状态 控制流的错误 数据的处理/解释错误 竞争状态 负荷状态 版本 文档

7.4.4 原因-结果图

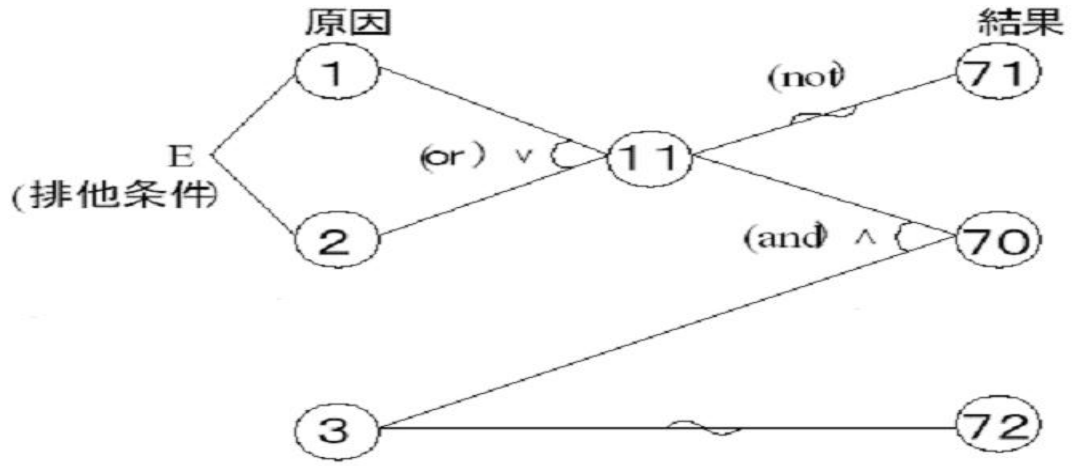
- 将程序的输入条件，环境条件等的原因，和处理，输出等的结果的逻辑关系，变换为逻辑图，从而作成判定表，进而设计测试项目。

原因-结果图的例(1)

- 设计：

  第1列的文字是‘A’或‘B’，第2列的文字是数字。
  满足这种条件时更新文件。
  第1列的文字不正确的话，打印X12的message。
  第2列的文字不正确的话，打印X13的message 。

原因-结果图的例(2)



① 第1列的文字是 'A'

② 第1列的文字是 'B'

③ 第2列的文字是 数字

70 更新文件

71 打印X12的message

72打印X13的message

原因-结果图的例(3)

使用原因结果图作成判定表

1.从图中选出1个结果。

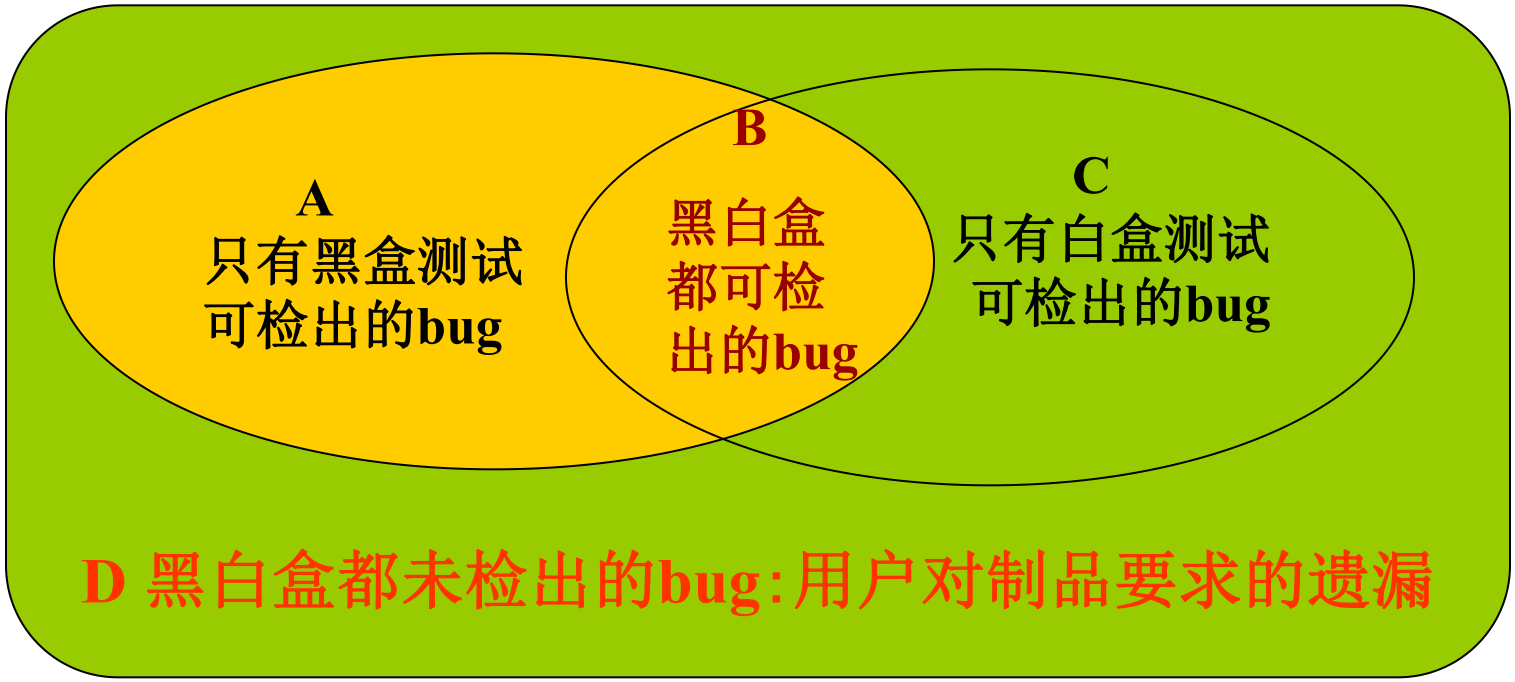
2. 分析所有可能获得这个结果的原因。

3.作成原因的组合表。

4.分析所有的结果

测试项 / 原因结果		1	2	3	4
1	第**1**列的文字是‘**A**’	1	0	0	
2	第**1**列的文字是‘**B**’	0	1	0	
3	第**2**列的文字是 数字	1	1		0
70	更新文件	1	1		
71	打印**X12**的**message**			1	
72	打印**X13**的**message**				1

白盒测试和黑盒子测试的关系



A＋B＋C＋D ＝软件全体的bug

测试技法的使用

■基本

等价类法，边界值法（＋代表值），错误推测法

■原因结果图法

复数个输入，复杂的输出和依存关系。

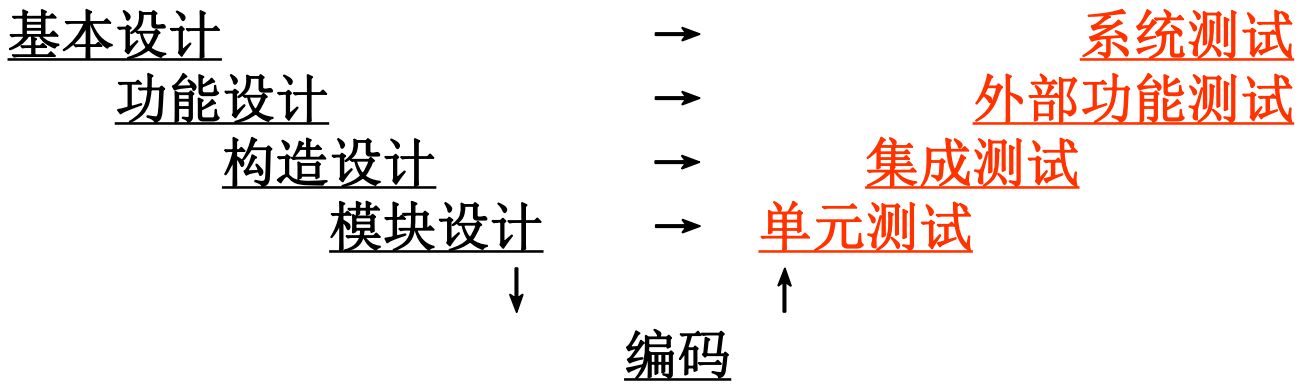
■状态迁移法

处理内部有状态迁移的情况

7.5 测试步骤
–软件测试策略

开发工程和测试的对应关系

V字型的测试工程

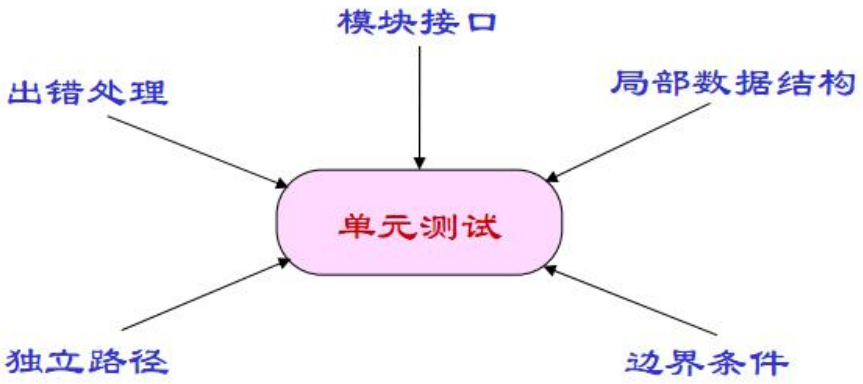


软件开发活动模型


需求阶段
设计阶段
实现阶段
测试阶段
验收阶段
用例情景测试
原型走查
模型走查
需求评审
模型走查
原型走查
设计评审
代码走查
接口分析
文档评估
制定测试计划
制定测试计划
测试设计
编写测试用例
制定测试过程
单元测试
制定测试过程
集成测试
系统测试
α 测试
β 测试
验收测试
回归测试，质量保证

7.6 单元测试

- 单元测试集中检测软件设计的最小单元-模块
- 主要使用白盒测试技术



单元测试项目设计

➢模块功能的测试项：

模块的功能，主要使用黑盒测试法设计测试项目

➢模块通路的测试项：

模块的详细实现和代码，主要使用白盒测试法设计测试项目

测试项目的设计—单元测试项目的设计

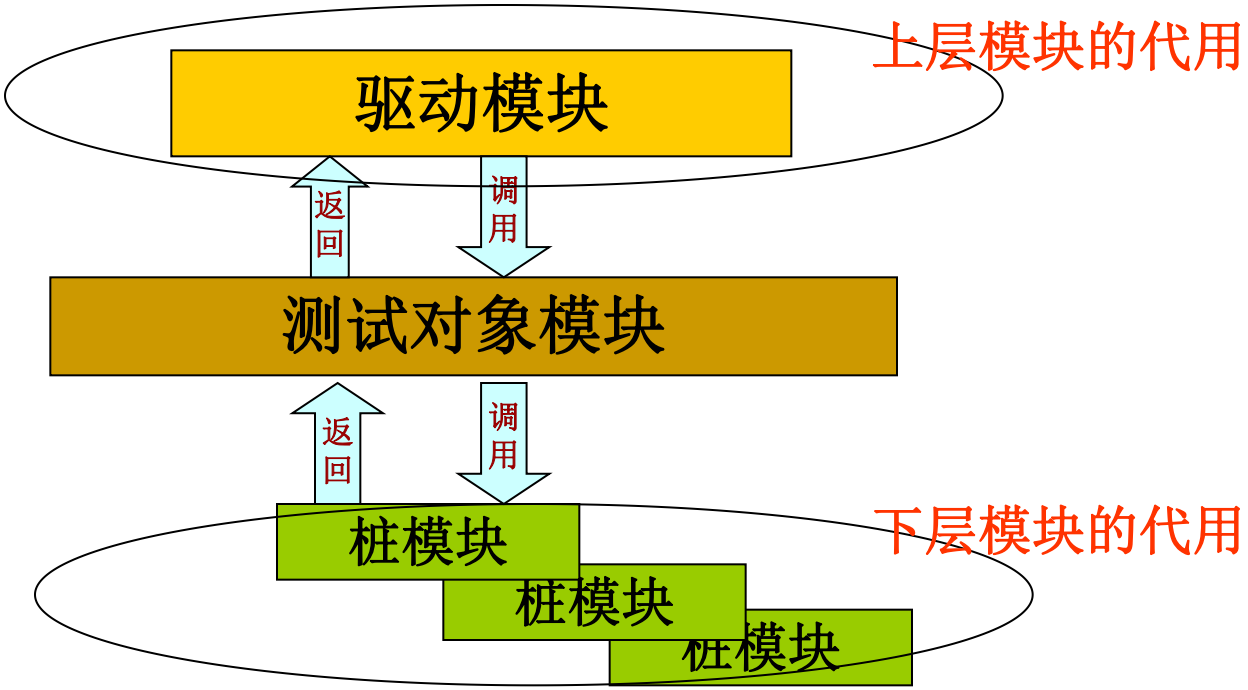
测试观点例（checklist）

- 是否能够确认所有的输入/输出条件，以及option(option可以理解为影响模块动作的全局变量，共享内存中的flag等)
- 是否能够确认边界值，中间值(MAX+1, MAX, MAX-1, MIN, MIN-1等)
- 是否能够确认所有的错误处理动作(所有的错误处理中内存释放，文件关闭，共享内存释放，status设置，errno保存，message输出等是否能够确认）
- 所有的分支条件是否测试了TRUE和FALSE两种情况(if条件无论有无else分支，都必须测试两种情况)
- 所有新规和改造的源代码是否都覆盖到了
- 是否能够确认循环处理是否正确（循环0次，循环1次，循环最大次的情况全部需要测试，另外，在循环结束后对循环变量的引用也必须测试）
- 能够确认调用其他模块时的输入输出条件是否和设计一致(调用该模块的条件是否满足; 调用后的返回值处理是否妥当; 返回status, errno处理是否妥当; 该模块内部是否会分配内存; 从而需要在调用后释放；其他资源有无类似情况（文件，共享内存，socket等))
- 对于所有的共享内存结构，电文结构等，是否有测试其长度的测试项目(设计测试项目时，必须根据式样书设计，不能根据程序设计)
- 在bug修正后追加的UT中，是否能够确认该bug已经被修正

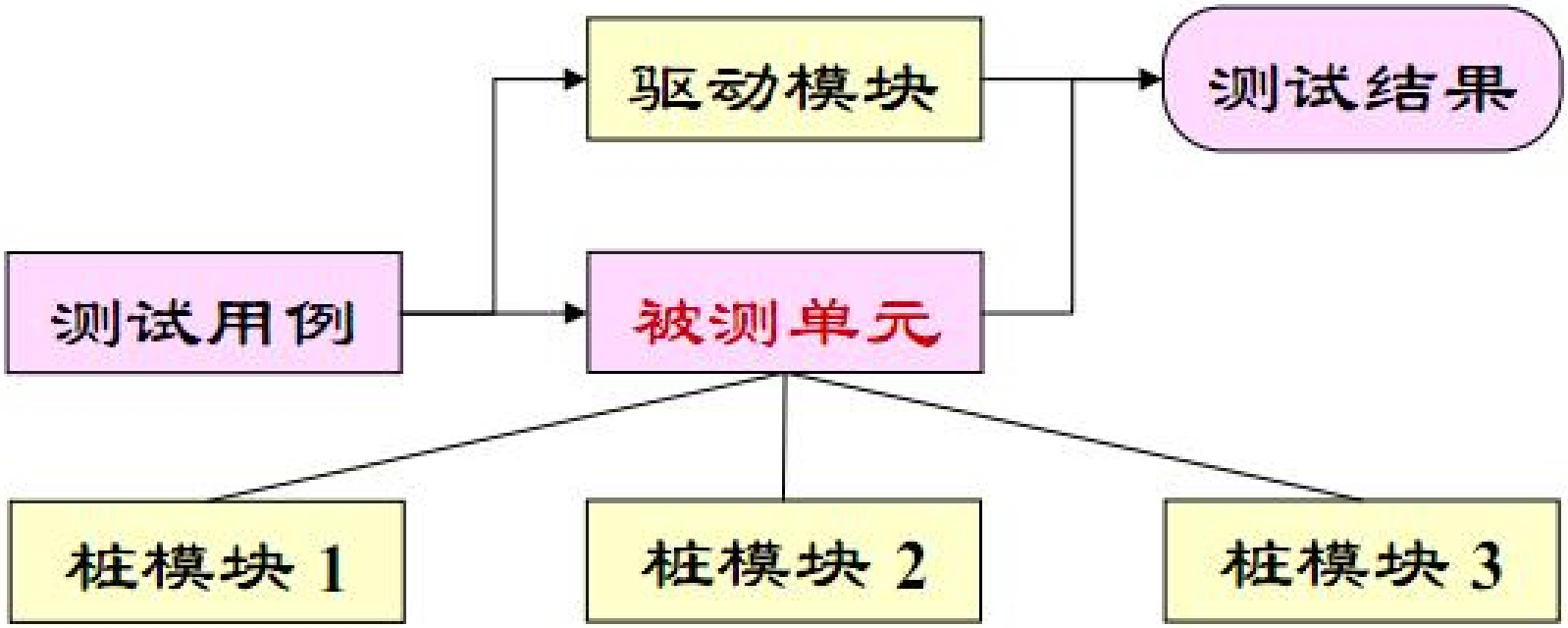
模块并不是一个独立的程序，要运行它就必须为其开发驱动软件和(或)存根（桩）软件。

驱动模块（Driver）：调要侧模块的代称

桩模块（Stub）　　：被调用侧的模块的代称



单元测试环境



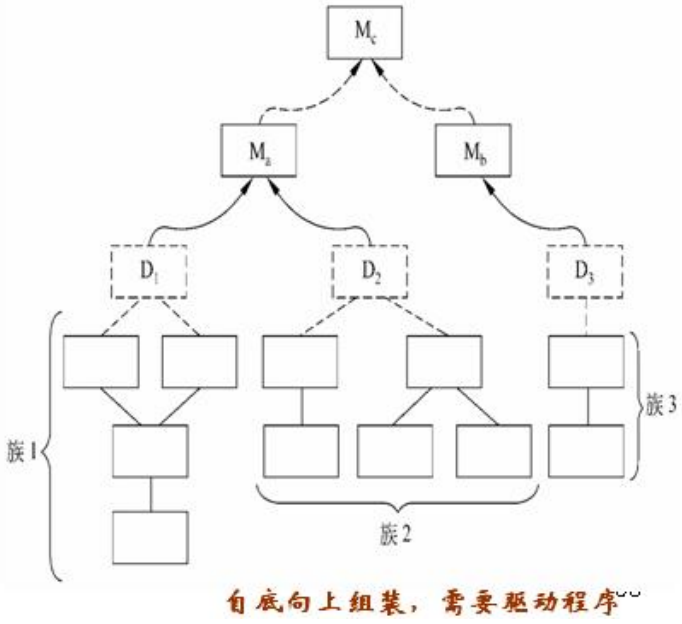
7.7 集成测试

- 集成测试　是测试和组装软件的系统化技术，其主要目标是发现与接口有关的问题。

  - 集成测试策略
    - 基于层次的集成：自顶上下与自底向上
    - 基于功能的集成：按照功能的优先级逐步将模块加入系统中
    - 基于进度的集成：把最早可获得的代码进行集成
    - 基于使用的集成：通过类的使用关系进行集成

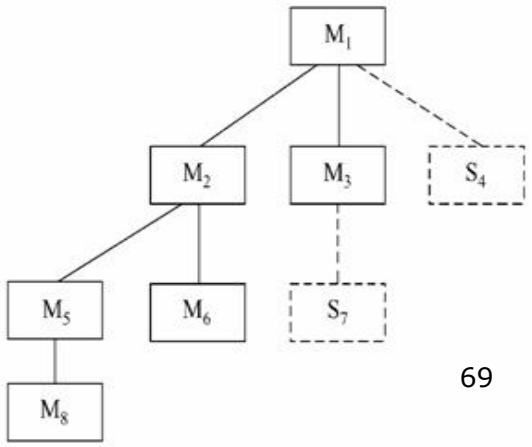
自底向上和顶向下的选择

- 自底向上（BottomUp）
  最下面的模块开始进行测试
  需要制作驱动模块
  然后测试上面的模块，不用制作桩模块
  而是使用已有的下层模块



自底向上组装，需要驱动程序

- 自顶向下（TopDown）
  最上面的模块开始进行测试
  需要制作桩模块
  然后测试下面的实际模块，不用制作 驱动模块



非渐增式与渐增式测试

1、非渐增式测试方法，即：先分别测试每个模块，再把所有模块按设计要求放在一起结合成所要的程序进行测试。

2、渐增式测试，即：先把下一个要测试的模块同已经测试好的那些模块结合起来进行测试，测试完以后再把下一个应该测试的模块结合进来测试。

这种每次增加一个模块的方法实际上同时完成单元测试和集成测试.

目前在进行集成测试时普遍采用渐增式测试方法。

测试项目的设计—集成测试项目的设计

- 由模块关联图获得测试项目
- 在单元测试项目中，有关模块间的接口的条目也使用
- 着眼于关联的所有模块间的接口

测试项目的设计—集成测试项目的设计

测试观点例

- 是否能够确认调用参数是否正常(参数个数，参数顺序，参数类型等是否正确）
- 是否能够确认错误处理时的动作是否正常(和UT中的3类似，但是这里是针对调用其他模块后的处理)
- 模块间的interface是否能够确认(比如，模块之间通过共享内存，全局变量，数据文件，电文结构进行接口，对这些数据的每一个field都要进行确认)
- 构成程序/component的所有模块间的调用通路是否都能确认(根据模块关联图/模块调用关系图)
- 如果调用时传入的参数和意图不符，是否会产生误动作

7.8 确认测试（系统测试）

- **1.** 功能测试（**Function Testing**） **usecase** 场景
- **2.** 负荷测试（**Stress Testing**） 用户量大 使用不同的场景
- **3.** 大容量测试（**Volume Testing**）
- **4.** 存储量测试（**Storage Testing**）
- **5.** 安全性测试（**Security Testing**）
- **6.** 性能测试（**Performance Testing**）
- **7.** 可靠性测试（**Reliablity Testing**）
- **8.** 恢复测试（**Recovery Testing**）
- **9.** 使用性测试（**Usabiliry Testing**）
- **10.**文档测试（**Documentation Testing**）
- **11.**工序测试（**Procedure Testing**）

测试项目的设计—功能测试项目的设计

- 由软件需求规格说明书，系统构成图、使用说明书等获得测试项目。

- 确认测试通常使用黑盒测试法。

测试项目的设计—功能测试项目的设计

测试观点例

- 操作上，接口上与设计相符
- 结果（例如画面构图等）与设计相符
- 操作的组合和接口的组合
- 错误**message**与设计相符
- 功能的组合
- 不只限于正常的使用方法，使用者错误的使用方法也进行推测设计

验收测试

- 验收测试（*Acceptance Testing*）
  - 验收测试是以用户为主的测试，一般使用用户环境中的实际数据进行测试。
  - 在测试过程中，除了考虑软件的功能和性能外，还应对软件的兼容性、可维护性、错误的恢复功能等进行确认。
- α测试与β测试
  - α测试与β测试是产品在正式发布前经常进行的两种测试；
    - α测试是由用户在开发环境下进行的测试；
    - β测试是由软件的多个用户在实际使用环境下进行的测试。

回归测试

- 回归测试（*Regression Testing*）
  - 回归测试是验证对系统的变更没有影响以前的功能，并且保证当前功能的变更是正确的。
  - 回归测试可以发生在软件测试的任何阶段，包括单元测试、集成测试和系统测试，其令人烦恼的原因在于频繁的重复性劳动。
  - 回归测试应考虑的因素
    - 范围：有选择地执行以前的测试用例；
    - 自动化：测试程序的自动执行和自动配置、测试用例的管理和自动输入、测试结果的自动采集和比较、测试结论的自动输出。

测试方法的分类总结

	设计的基础	特征	
		优点	缺点
白盒法 (只用于UT,IT)	程序的内部设计 ▪模块的内部构造 ▪程序开发者所见的设计	内部设计的网罗测试 ▪执行程序的所有逻辑通路的测试	外部/功能设计有，但是没有被实现的部分发生测试遗漏
黑盒法	程序的外部设计 ▪模块的功能 ▪用户所见的设计	从用户的立场出发的测试 ▪可能具有的所有输入值的组合测试	外部/功能设计设计没有表现的内部设计上的特殊处理发生测试遗漏

测试计划

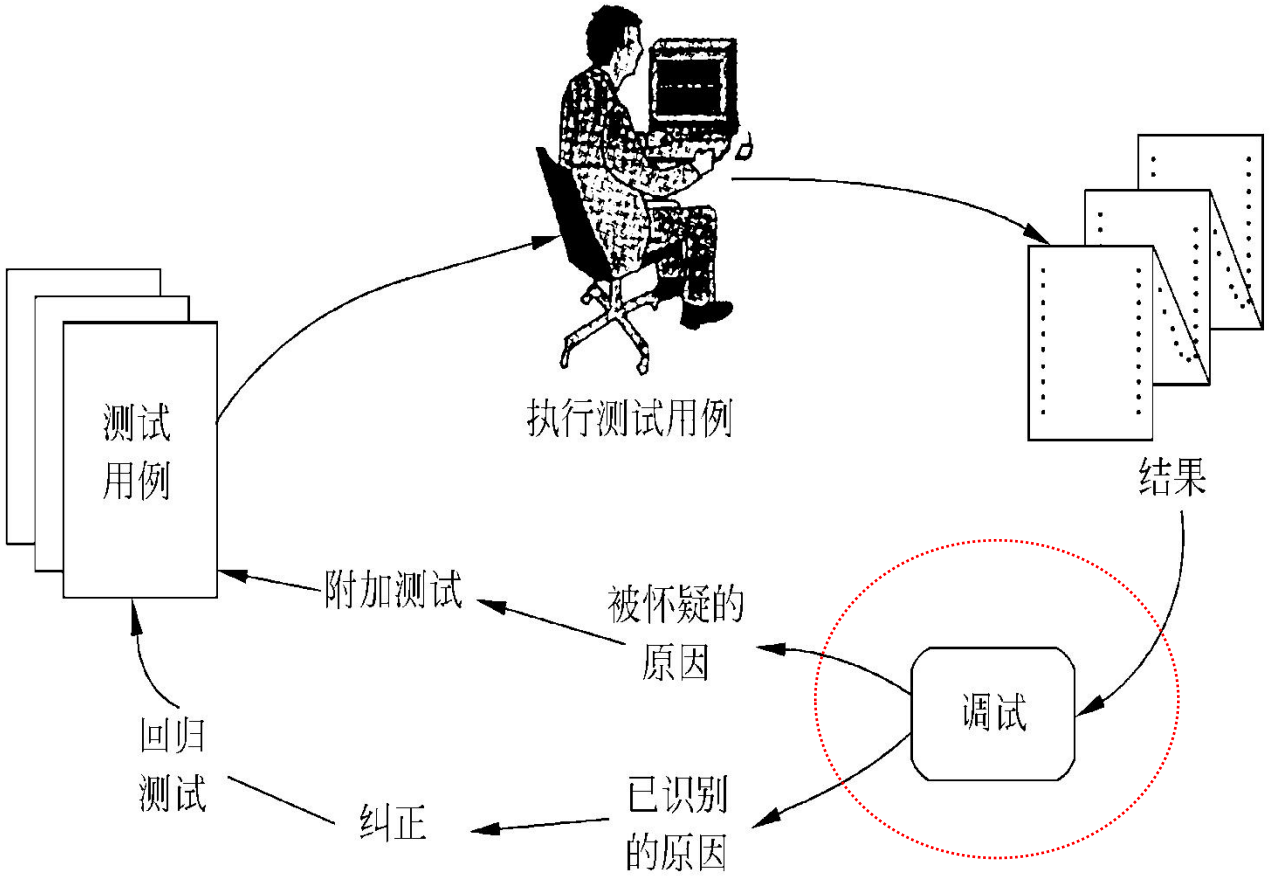
- 测试之前应该制定计划
- 测试过程中作好测试记录
- 排错　　**fixed code**

  修改所涉及的代码要尽可能地少
  不要随便修改/删除无把握的代码
  尽量不要修改底层的公共代码，例如基类
  修改完代码之后要追加测试项

- 交付之前 尽量少的修改代码
- 功能测试　系统（压力性能场景）　交付测试

7.9 调试

- 调试是在测试发现错误之后排除错误的过程。



7.9 调试

- 方法：

  输出储存器内容
  打印语句
  自动工具

- 核心：

  找到出错的原因，寻找解决办法。

调试策略

- 试探法
- 回朔法
- 对分法
- 询问他人寻求帮助。

7.10 软件可靠性

- 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。

小结

- 测试包含静态测试和动态测试，测试步骤至少分为：
  - 1. 模块测试 --- 单元
  - 2. 子系统测试 --- 局部
  - 3. 系统测试 --- 集成
  - 4. 验收测试 --- 用户参与
- 白盒测试和黑盒测试是软件测试的两类基本方法
- 调试的方法

谢谢！

END.