中国石油大学（华东）
CHINA UNIVERSITY OF PETROLEUM

软件工程

主要内容

第一章 软件工程学概述
第二章 可行性研究
第三章 需求分析
第四章 总体设计
第五章 详细设计
第六章 编码与测试
第七章 软件维护
第八章 面向对象方法学
第九章 面向对象分析设计与实现
第十章 软件项目管理

第六章　系统实现

第一节　编码

第二节　软件测试基础

第三节　单元测试

第四节　集成测试

第五节　确认测试

第六节　平行运行

第七节　软件测试技术

第八节　软件调试

第九节　软件可靠性

第六章　系统实现

通常把编码和测试统称为系统实现。

- 编码就是把软件设计结果翻译成用某种程序设计语言书写的程序。
- 测试是在软件投入生产性运行之前，尽可能多地发现软件中的错误，是保证软件质量的关键步骤。

第一节　编码

一、选择程序设计语言

程序设计语言是人和计算机通信的最基本的工具，它的特点必然会影响人的思维和解题方式，会影响人和计算机通信的方式和质量，也会影响其他人阅读和理解程序的难易程度。因此，编码之前的一项重要工作就是选择一种适当的程序设计语言。

1. 选择的主要实用标准：
   - 系统用户的要求
   - 可以使用的编译程序
   - 可以得到的软件工具
   - 工程规模
   - 程序员的知识
   - 软件可移植性要求
   - 软件的应用领域

2．程序设计风格

程序实际上也是一种供人阅读的文章，有一个文章的风格问题。应该使程序具有良好的风格。

- 程序内部的文档
- 数据说明
- 语句结构
- 输入／输出方法

3. 效率

效率主要是指处理机时间和存储器容量两个方面。效率属于性能要求，软件应该像对它要求的那样有效，而不应该如同人类可能做到的那样有效。

- 程序运行时间
- 存储器效率
- 输入／输出效率

程序的效率和程序的简单程度是一致的，不要牺牲程序的清晰性和可读性来不必要地提高效率。

第二节　软件测试基础

一、软件测试的目的和重要性

因为开发工作的前期不可避免地会引入错误，测试的目的是为了发现和改正错误，这对于某些涉及人的生命安全或重要的军事、经济目标的项目显得尤其重要。

- 1963 年美国飞往火星的火箭爆炸，原因是 FORTRAN 程序：

DO 5 I=1，3 误写为：DO 5 I=1. 3　损失数千万美元。

- 1967 年苏联“联盟一号”宇宙飞船返回时因忽略一个小数点，在进入大气层时打不开降落伞而烧毁。

二、软件测试的特点

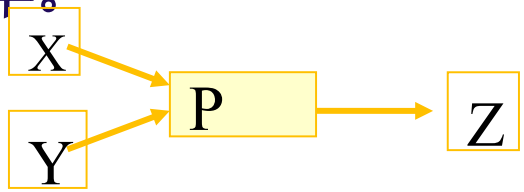
1. 软件测试的开销大

按照 Boehm 的统计，软件测试的开销大约占总成本的 30%-50% 。例如： APPOLLO 登月计划， 80% 的经费用于软件测试。

2. 不能进行“穷举”测试

只有将所有可能的情况都测试到，才有可能检查出所有的错误。但这是不可能的。

- 例：程序 P 有两个整型输入量 X 、 Y ，输出量为 Z ，在 32 位机上运行。所有的测试数据组（ X_i ， Y_i ）的数目为： $2^{32} * 2^{32} = 2^{64}$ 。若执行需 10^{-3} 秒，测试大约需用一万年。



3. 软件测试难度大

根据上述分析，既然不能进行 “穷举”测试，又要查出尽可能多的错误，软件测试工作的难度大。只有选择 —

“高效的测试用例”

- 什么是“高效的测试用例”？
- 如何选择“高效的测试用例”？

这就是本章讨论的主要问题！！！

三、软件测试的基本原则

1. 尽量不由程序设计者进行测试
2. 关键是注重测试用例的选择
   - 输入数据的组成（输入数据、预期的输出结果）
   - 既有合理输入数据，也有不合理的输入数据。
   - 用例既能检查应完成的任务，也能够检查不应该完成的任务。长期保存测试用例。
   -

三、软件测试的基本原则

3. 所有的测试都应当追溯到用户要求，导致程序不能满足用户要求的错误是严重错误

4. 充分注意测试中的群集现象

经验表明，测试发现的错误中的 80% 很可能出自 20% 的模块，换句话说，测试后程序中残存的错误数目与该程序中已发现的错误数目成正比。

5. 妥善保存测试计划、测试用例、出错统计和最终分析报告，为维护提供方便

四、软件测试方法

1. **黑盒测试**

   确信对每个输入，观察到输出与期望的输出是否匹配。

   —— 对功能测试

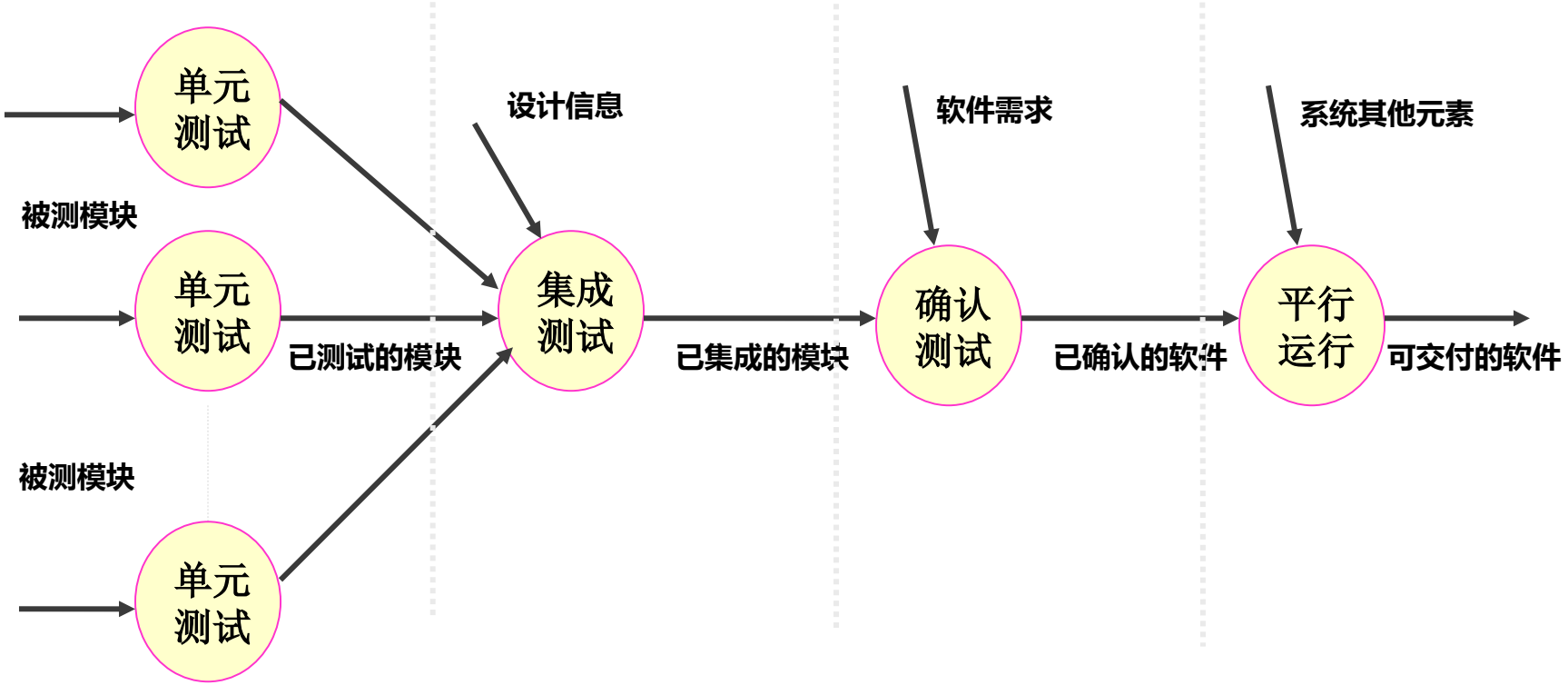
2. **白盒测试**

   利用程序的结构进行测试。

   —— 对结构测试

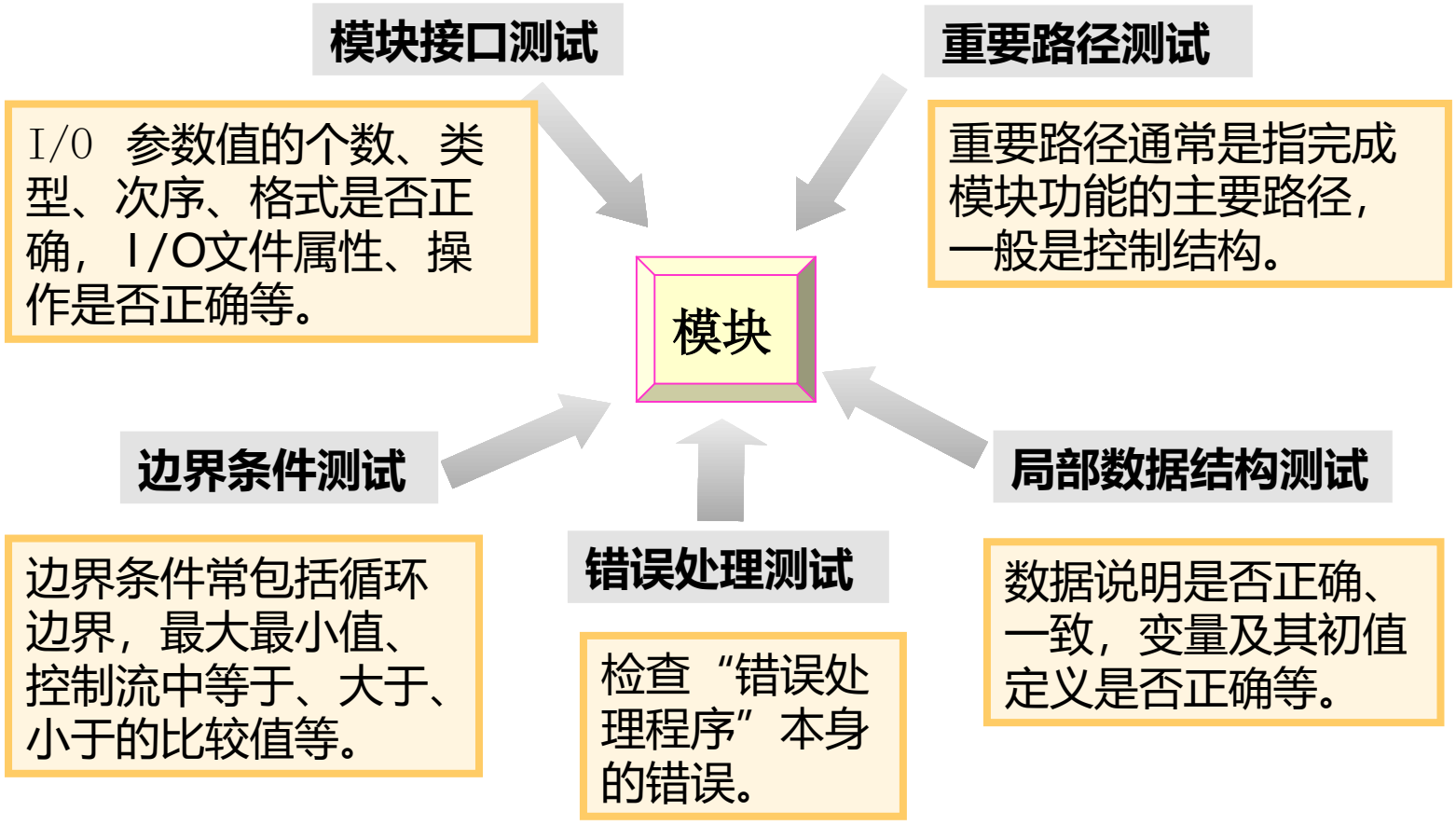
五、测试步骤及策略

所有测试过程都应采用综合测试策略；即先作静态分析，再作动态测试。并事先制订测试计划。测试过程通常可分 4 步进行：



第三节　单元测试

一、测试内容



二、测试步骤

1. 代码审查

人工测试程序可以由编写者本人非正式地进行，也可以由审查小组正式进行；

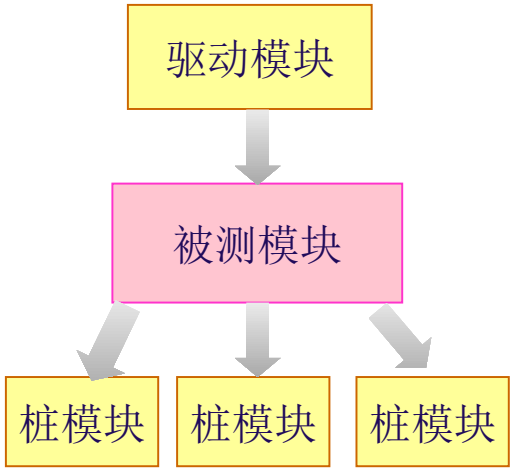
审查小组最好由四人组成：

- 组长：有能力的程序员、没有直接参与这项工程；
- 程序的设计者；
- 程序的编写者；
- 程序的测试者。

2. 计算机测试

考虑到被测模块与其它模块的联系，因此测试时需要使用两类辅助模块来模拟其他模块。

- 驱动模块（ driver ）— 模拟主程序功能，用于向被测模块传递数据，接收、打印从被测模块返回的数据。
- 桩模块（ stub ）— 又称为假模块，用于模拟那些由被测模块所调用的下属模块功能。程序的编写者；



第四节　集成测试

集成测试也称为联合测试或组装测试，重点测试模块的接口部分，需设计测试过程使用的驱动模块或桩模块。

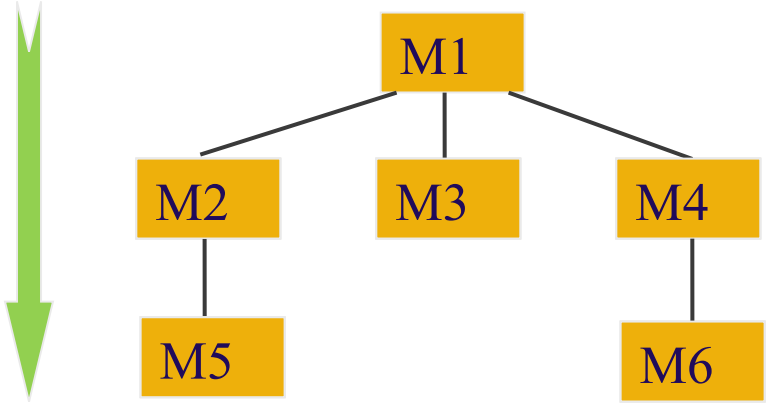
一、集成测试的任务

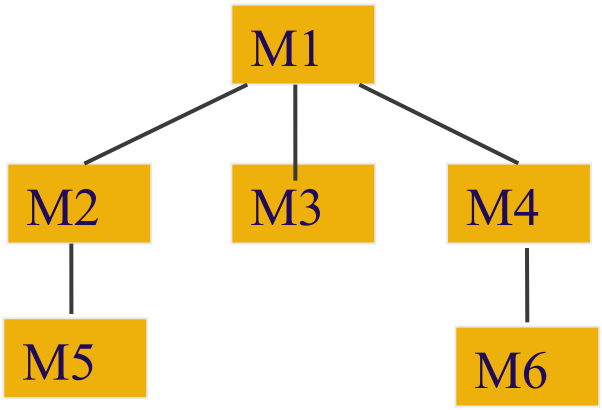
- 确定模块组装方案，将经过测试的模块组装为一个完整的系统。组装方案分为渐增式及非渐增式。
- 测试方法以黑盒法为主，按照组装方案进行测试。

非渐增式：先测试每个模块，再把模块组合成程序。

渐增式：先进行模块测试，然后将这些模块逐步组装成较大的系统，每连接一个模块进行一次测试。渐增式测试有 2 种模式：自顶而下集成和自底而上集成。

1. 自顶而下集成





程序模块示意图

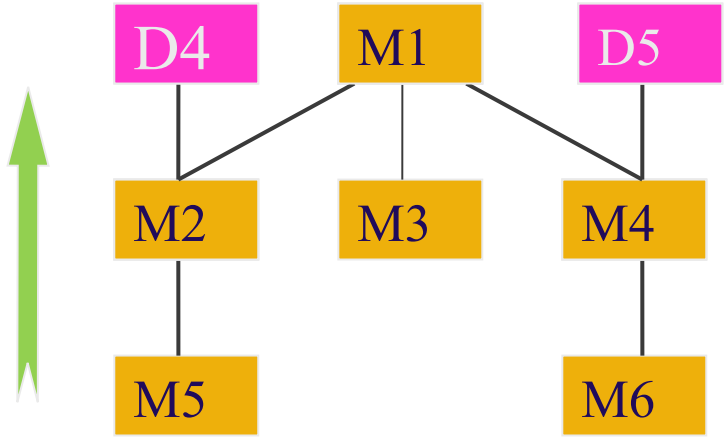
第一步，测试主控模块 M1，设计桩模块 S1、S2、S3，模拟被 M1 调用的 M2、M3、M4。

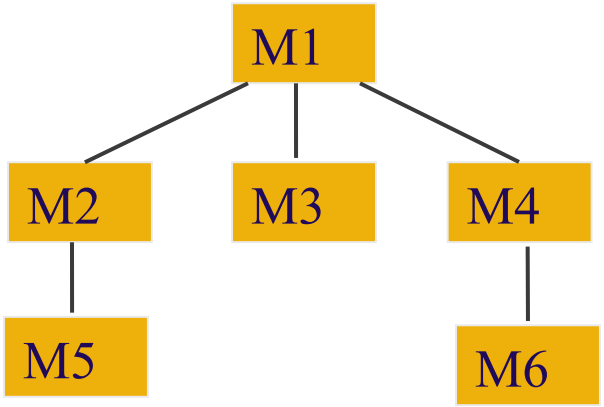
第二步，依次用 M2、M3、M4 替代桩模块 S1、S2、S3，每替代一次进行一次测试。

第三步，对由主控模块 M1 和模块 M2、M3、M4 构成的子系统进行测试，设计桩模块 S4、S5。

第四步，依次用模块 M5 和 M6 替代桩模块 S4、S5，并同时进行新的测试。组装测试完毕。

2. 自底而上集成





程序模块示意图

第一步，对最底层的模块 M3 、 M5 、 M6 进行测试，设计驱动模块 D1 、 D2 、 D3 来模拟调用。

第二步，用实际模块 M2 、 M1 和 M4 替换驱动模块 D1 、 D2 、 D3 。

第三步，设计驱动模块 D4 、 D5 模拟调用，分别对新子系统进行测试。

第四步，把已测试的子系统按程序结构连接起来完成程序整体的组装测试。

两种策略的比较

- “自顶向下”法的主要优点：不需要测试驱动程序，能够在测试阶段的早期实现并验证系统的主要功能，而且能在早期发现上层模块的接口错误；
- “自顶向下”法的主要缺点：需要桩模块程序，可能遇到与此相联系的测试困难，低层关键模块中的错误发现较晚，而且用这种方法在早期不能充分展开人力；
- “自底向上”法的优缺点与“自顶向下”法刚好相反。

3. 混合策略（“三明治”方法）

在具体测试中，采用混合策略。

① 改进的“自顶向下”法：

基本使用“自顶向下”法，但在测试早期，使用“自底向上”法测试少数的关键模块；

② 混合法：

对软件结构中较上层，使用的是“自顶向下”法；对软件结构中较下层，使用的是“自底向上”法，两者相结合。

集成过程的原则：

① 尽早测试关键模块。

② 尽早测试包含 I/O 的模块。

趣事

业的“潜规则”？

里，这个原版程序，里面有183个bug。

无修复它们。”

修复了13个。”

序，剩下170个继续努力。”

现在它有264个bug了。”

二、回归测试

回归测试是指重新执行已经做过的测试的某个子集，以保证修改变化没有带来非预期的副作用。

- 因为在集成测试过程中每当一个新模块结合进来时，程序就发生了变化：建立了新的数据流路径，可能出现了新的 I/O 操作，激活了新的控制逻辑。这些变化有可能使原来工作正常的功能出现问题。

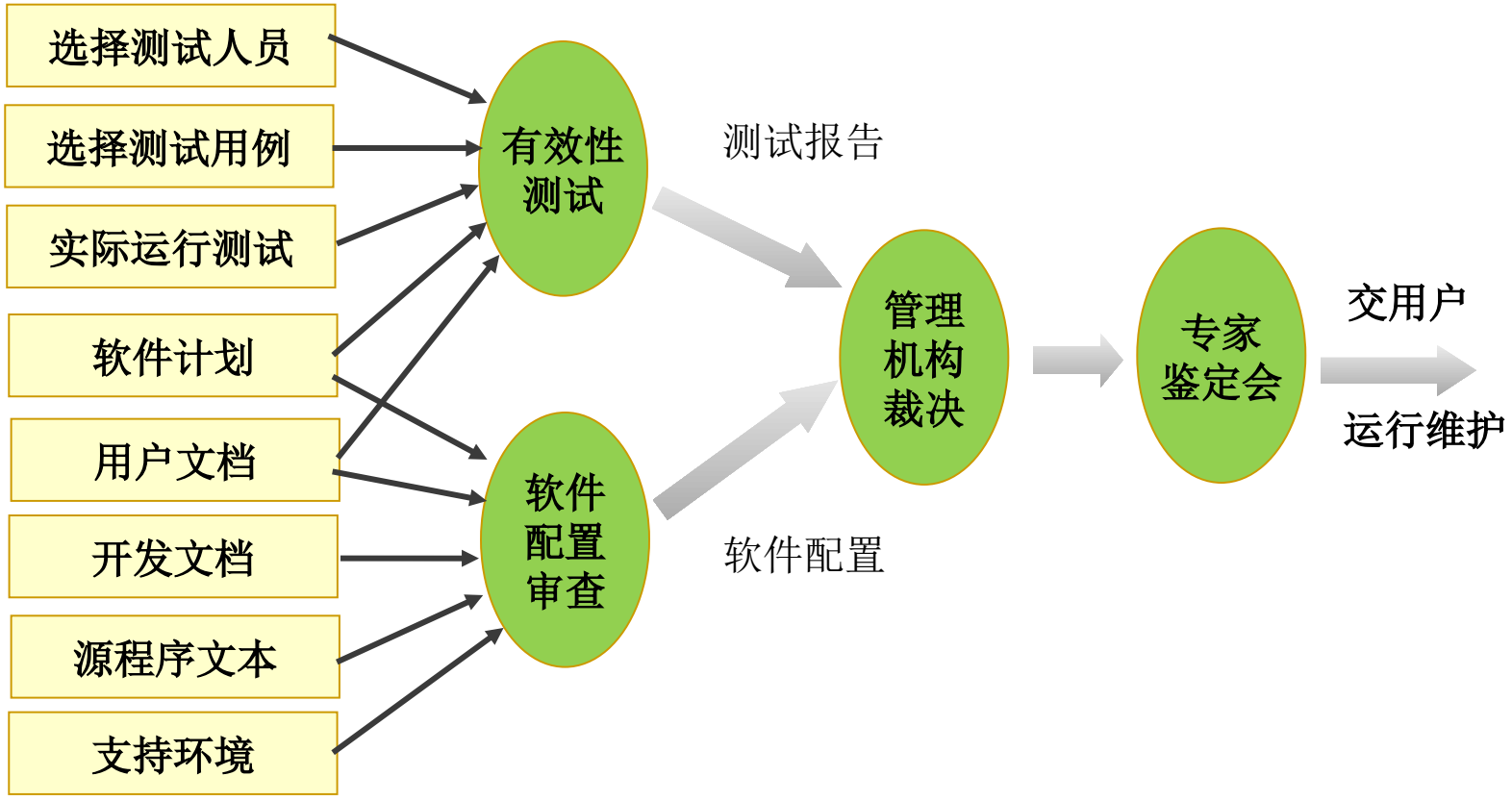
- 回归测试的作用就是用于保证由于调试或其他原因引起的变化，不会导致非预期的软件行为或额外错误的测试活动。

回归测试集（已执行过的测试用例的子集）包括下述 3 类不同的测试用例：

- 检测软件全部功能的代表性测试用例；
- 专门针对可能受修改影响的软件功能的附加测试；
- 针对被修改过的软件成分的测试。

第五节　确认测试

确认测试又称验收测试，其任务是验证系统的功能、性能等特性是否符合需求规格说明。（验证软件有效性）



一、确认测试步骤

1. 有效性测试

制定测试计划和测试过程，运用黑盒法，验证软件特性是否与需求符合。

2. 软件配置复查

软件配置是指软件工程过程中所产生的所有信息项：文档、报告、程序、表格、数据。随着软件工程过程的进展软件配置项快速增加和变化，应复查软件配置项是否齐全、一致。

3. □测试和□测试

如果软件是专为某个客户开发的，则可进行一系列验收测试，以便用户确认所有需求都得到了满足；如果一个软件是为许多客户开发的，则需采用　　测试和　　测试。

- □ 测试：是在开发机构的指导下，在开发者的场所，由个别用户在确认测试阶段后期对软件进行测试，目的是评价软件的FLURPS （功能、局域化、可用性、可靠性、性能和支持），注重界面和特色。
- □ 测试：由支持软件预发行的客户，在一个或多个客户场所，对 FLURPS 进行测试，主要目的是测试系统的可支持性。

Function Testing　功能测试
Usability Testing　可使用性测试
Performance Testing　性能测试
Local Area Testing　局域化测试
Reliability Testing　可靠性测试
Supportability Testing　可支持性测试

第六节　平行运行

所谓平行运行就是同时运行新开发出来的系统和将被它取代的旧系统，以便比较新旧两个系统的处理结果。这样做的具体目的有如下几点：

- 可以在准生产环境中运行新系统而又不冒风险；
- 用户能有一段熟悉新系统的时间；
- 可以验证用户指南和使用手册之类的文档；
- 能够以准生产模式对新系统进行全负荷测试，可以用测试结果验证性能指标。

如　何　测　试　？

- 关键　----　设计测试方案。
- 测试方案　----　包括：具体的测试目的，应该输入的测试数据和预期的结果。
- 通常又把测试数据和预期的输出结果称为测试用例。其中最困难的问题是设计测试用的输入数据。
- 不同的测试数据发现程序错误的能力差别很大，为了提高测试效率降低测试成本，应该选用高效的测试数据。

第七节　软件测试技术

软件测试技术分为两类：白盒测试技术、黑盒测试技术。

- 白盒测试技术

分析程序的内部逻辑结构，注意选择适当的覆盖标准，设计测试用例，对主要路径进行尽可能多的测试。

- 黑盒测试技术

不考虑程序的内部结构与特性，只根据程序功能或程序的外部特性设计测试用例。

一、白盒测试技术

白盒法又称为逻辑覆盖法，其测试用例选择，是按照不同覆盖标准确定的。

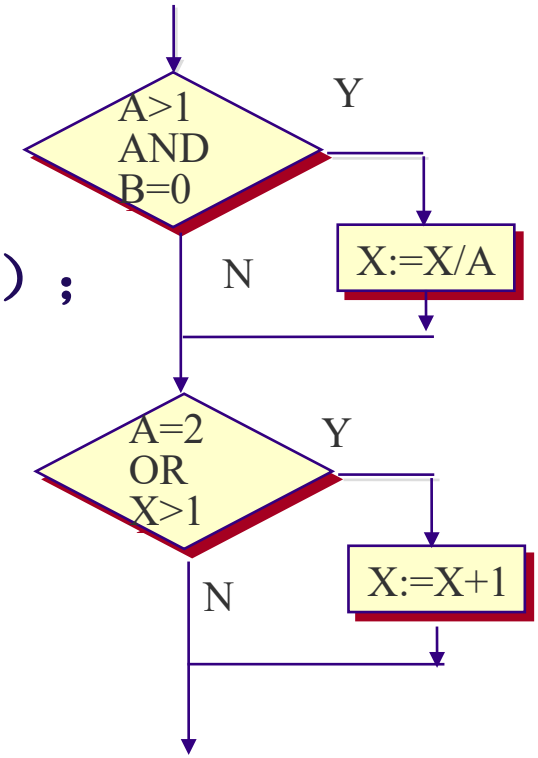
发现错误的能力	标　准	含　　义
1	语句覆盖	每条语句至少执行一次
2	判定覆盖	每一判定的每个分支至少执行一次
3	条件覆盖	每一判定中的每个条件，分别按“真”、“假”至少各执行一次
4	判定／条件覆盖	同时满足判定覆盖和条件覆盖的要求
5	条件组合覆盖	求出判定中所有条件的各种可能组合值，每一可能的条件组合至少执行一次

白盒法步骤：

- 选择逻辑覆盖标准；
- 按照覆盖标准列出所有情况；
- 选择确定测试用例；
- 验证分析运行结果与预期结果。

例：用白盒法测试以下程序段

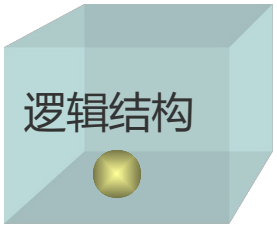
```
Procedure （ VAR A， B， X： REAL ）；
BEGIN
  IF  （ A>1） AND （B=0）
        THEN  X:=X/A ；
  IF （A=2）  OR  （X>1）
        THEN   X:=X+1
END;
```

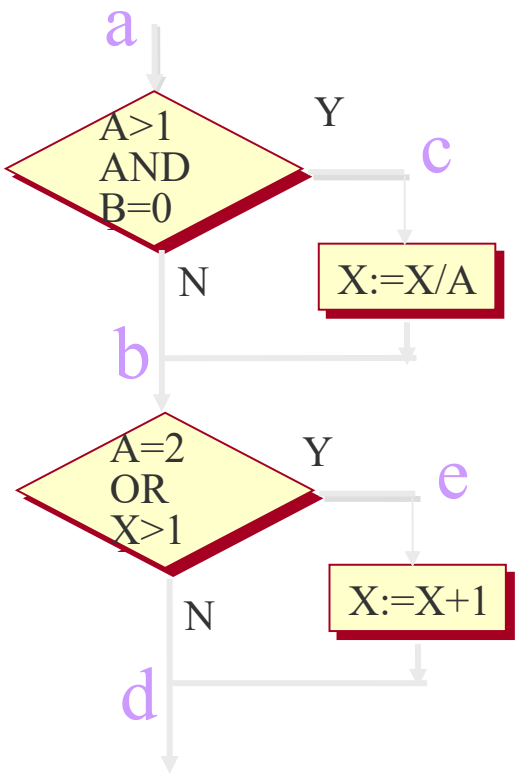


1. 基于逻辑覆盖测试

（1）语句覆盖

- 使得程序中每个语句至少都能被执行一次。





满足语句覆盖的情况：
执行路径： ace

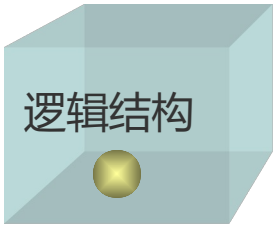
用例格式：
[输入 (A, B, X) ， 输出 (A, B, X)]

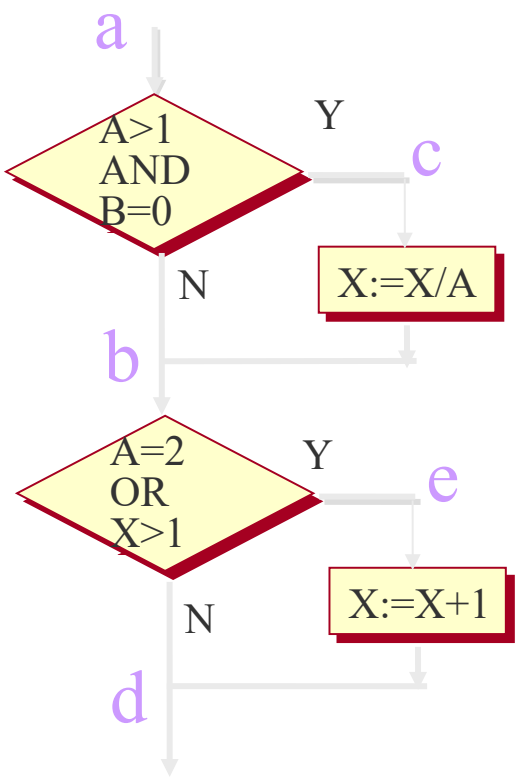
选择用例：
[(2, 0, 4) , (2, 0, 3)]

1. 基于逻辑覆盖测试

（2）判定覆盖

- 使得程序中每个判定至少为 TRUE 或 FALSE 各一次。





覆盖情况：应执行路径
ace ∧ abd 或： abe ∧ acd

选择用例（其一）：

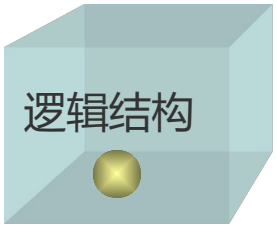
(1) [(2, 0, 4), (2, 0, 3)] ace
[(1, 1, 1), (1, 1, 1)] abd

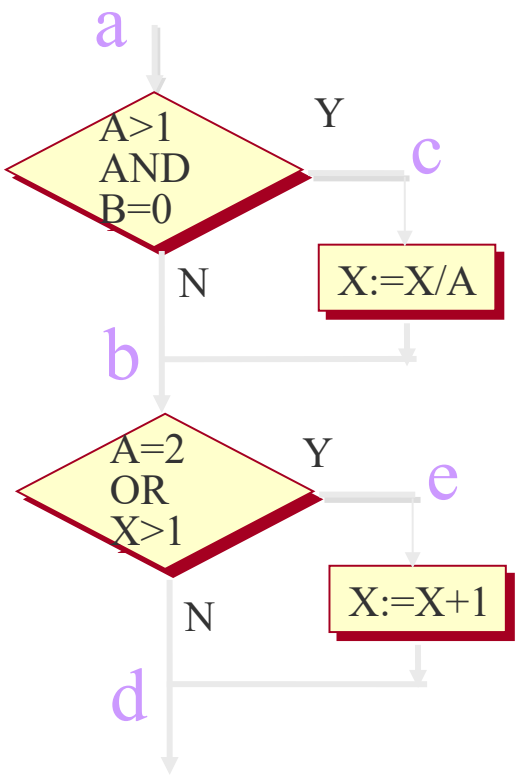
(2) [(1, 1, 2), (1, 1, 3)] abe
[(3, 0, 3), (3, 0, 1)] acd

1. 基于逻辑覆盖测试

（3）条件覆盖

- 使得判定中的每个条件获得各种可能的结果。





应满足以下覆盖情况：

判定一：A>1，　A≤1, B=0，　B≠0

判定二：A=2，　A≠2, X>1，　X≤1

选择用例：

[(2, 0, 4), (2, 0, 3)]　ace

[(1, 1, 1), (1, 1, 1)]　abd

比较：[(1, 0, 3), (1, 0, 4)]　abe

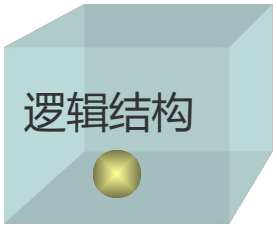
[(2, 1, 1), (2, 1, 2)]　abe

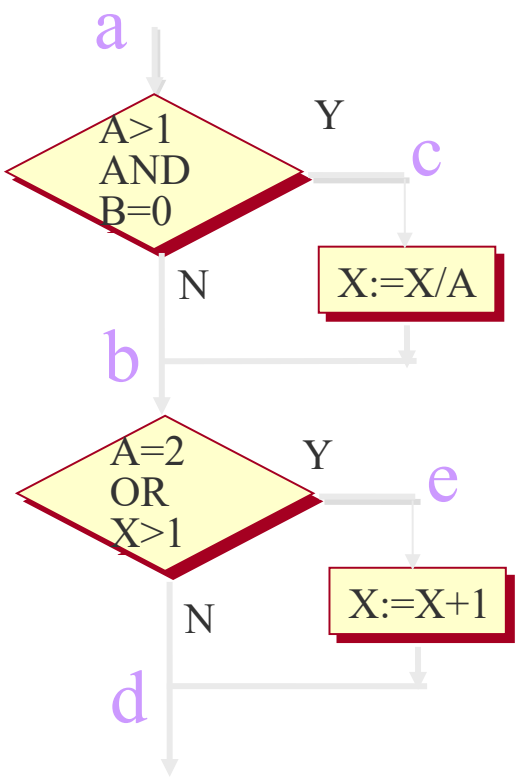
满足条件覆盖，但不满足判断覆盖。

1. 基于逻辑覆盖测试

（4）判定/条件覆盖

- 同时满足判断覆盖和条件覆盖。





应满足以下覆盖情况：

条件：　A>1，　A≤1，　B=0，　B≠ 0

　　A=2，　A≠ 2，　X>1，　X≤1

应执行路径：

ace ∧　abd　或：　acd ∧　abe

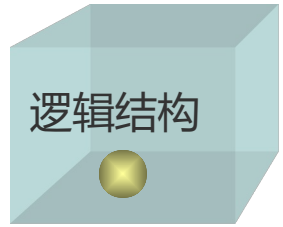
选择用例：

[(2, 0, 4), (2, 0, 3)]　(ace)

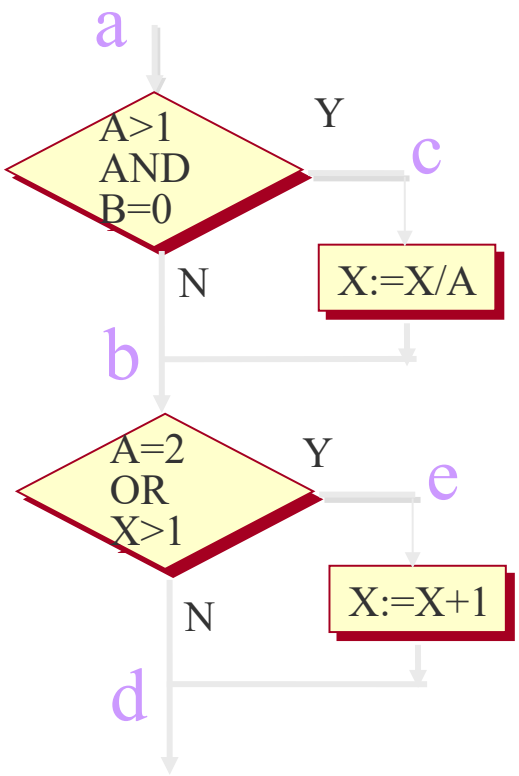
[(1, 1, 1), (1, 1, 1)]　(abd)

1. 基于逻辑覆盖测试

（5）条件组合覆盖



- 使得每个判定中条件的各种可能组合都至少出现一次。



满足以下覆盖情况：

① A>1, B =0　② A>1, B≠ 0

③ A≤1, B =0　④ A≤1, B≠ 0

⑤ A=2, X>1　⑥ A=2, X≤1

⑦ A≠ 2, X>1　⑧ A≠ 2, X≤1

选择用例：

[(2, 0, 4), (2, 0, 3)]　①　⑤

[(2, 1, 1), (2, 1, 2)]　②　⑥

[(1, 0, 3), (1, 0, 4)]　③　⑦

[(1, 1, 1), (1, 1, 1)]　④　⑧

2. 基于控制结构测试

（1）基于路径测试

是在流图的基础上，通过计算环形复杂度，导出基本路径集合，从而设计测试用例，　保证这些路径至少通过一次。

- 基本路径测试的步骤为：

1）根据过程设计的结果画出相应的流图

2）计算流图 G 的环形复杂度 V(G)

3）确定线性独立路径的基本集合

4）设计基本集合中每条路经的测试用例

2. 基于控制结构测试

（1）基于路径测试

是在流图的基础上，通过计算环形复杂度，导出基本路径集合，从而设计测试用例，　保证这些路径至少通过一次。

- 基本路径测试的步骤为：

1）根据过程设计的结果画出相应的流图

2）计算流图 G 的环形复杂度 V(G)

3）确定线性独立路径的基本集合

4）设计基本集合中每条路经的测试用例

2. 基于控制结构测试

（1）基于路径测试

独立路径是指包括一组以前没有处理的语句或条件的一条路径。用流图术语描述，一条独立路径是至少包含有一条在定义该路径之前不曾用过的边。

例如，在右图所示的流图中，

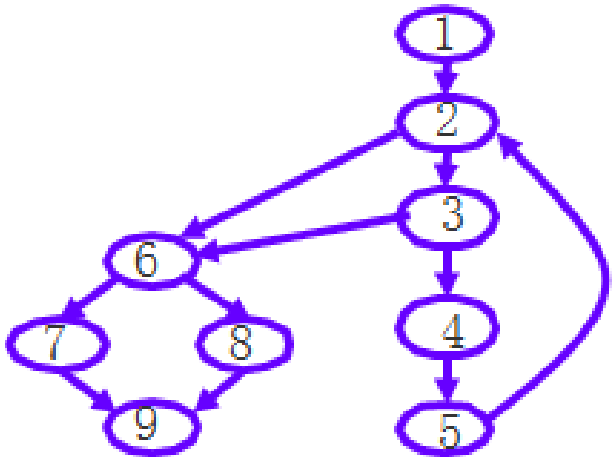
一组独立的路径是：

path1：　1-2-6-7-9

path2：　1-2-6-8-9

path3：　1-2-3-6-…

path4：　1-2-3-4-5-2-…



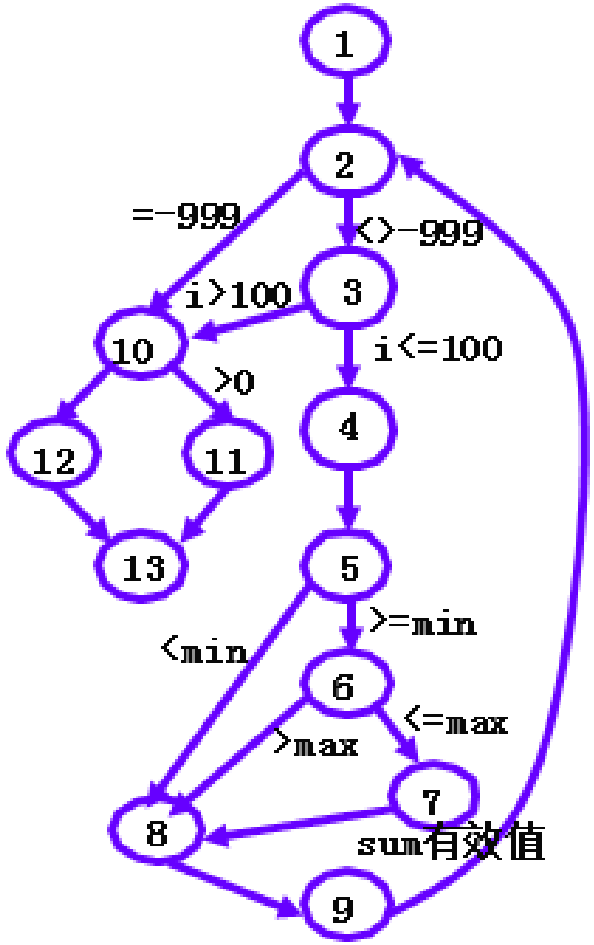
基本路径集不是唯一的，对于给定的程序图，可以得到不同的基本路径集。

例：计算不超过 100 个在规定值域内的有效数字的平均值

```
Procedure average
1:    i=1; total.input=total.valid=0; sum=0;
2:    do while value[i] <>-999
3:       and total.input<=100
4:    increment total.input by 1;
5:    if value[i]>=minimum
6:       and value[i]<=maximum
7:    then increment total.valid by 1;
          sum=sum+value[i];
8:    endif
      increment i by 1;
9:    enddo
10:   if total.valid>0
11:   then average=sum/total.valid;
12:   else average=-999
13:   end average
```

第一步画流图，如右图所示

```
Procedure average
1:    i=1; total.input=total.valid=0; sum=0;
2:    do while value[i] <>-999
3:       and total.input<=100
4:    increment total.input by 1;
5:    if value[i]>=minimum
6:       and value[i]<=maximum
7:    then increment total.valid by 1;
          sum=sum+value[i];
8:    endif
      increment i by 1;
9:    enddo
10:   if total.valid>0
11:   then average=sum/total.valid;
12:   else average=-999
13:   end average
```



第二步计算环形复杂度： V(G)=6

第三步确定独立路径集

路径

1：　1-2-10-11-13

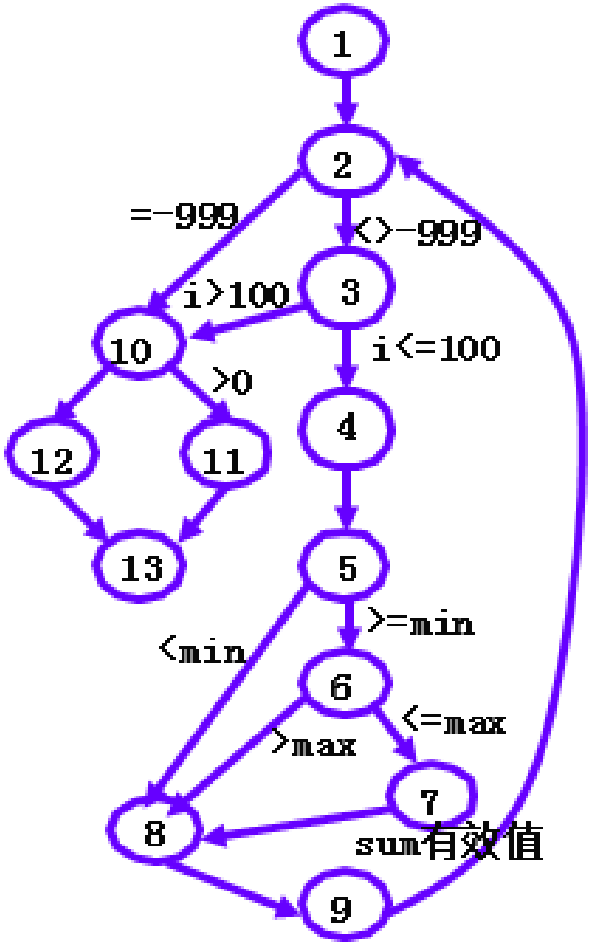
2：　1-2-10-12-13

3：　1-2-3-10-11-13

4：　1-2-3-4-5-8-9-2-•••

5：　1-2-3-4-5-6-8-9-2-•••

6：　1-2-3-4-5-6-7-8-9-2-•••



第四步 设计测试用例

输入	预期输出
1： k个有效输入 k<i 第i个=-999 2≤i≤100	基于k的正确平均值和总数 不能独立测试
2： Value(1)=-999	average=-999
3： 100个有效输入 i=100	前100个数的正确平均值 总数为100，不能独立测试
4: i个<最大值的输入， i<100 其中有k个>最小值 0<k<i	基于k的正确平均值和总数
5: i个>最小值的输入， i<100 其中有k个<最大值 0<k<i	基于k的正确平均值和总数
6: i个有效输入 0<i<100	正确的平均值和总数

2. 基于控制结构测试

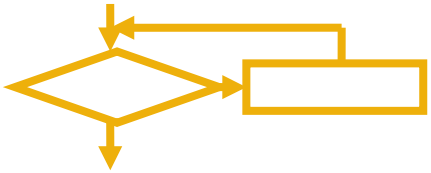
（2）循环测试

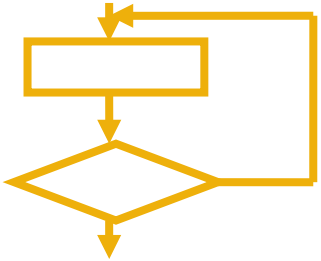
用于测试循环结构的有效性。在结构化程序中，循环通常有三类：简单循环、嵌套循环和串接循环。

- 简单循环

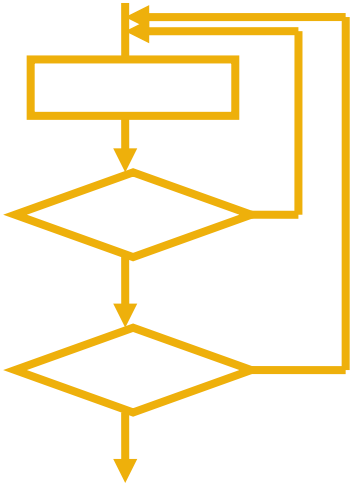
使用下列测试集，其中 n 为允许通过循环的最大次数。

- ✓跳过循环
- ✓只通过循环一次
- ✓通过循环两次
- ✓通过循环 m 次，其中 m<n-1
- ✓通过循环 n-1, n, n+1 次





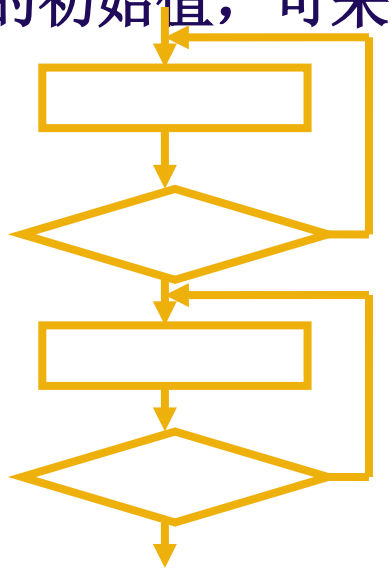
- 嵌套循环
  - ✓从内层循环开始测试，其它循环设为最小值。
  - ✓对内层循环使用简单循环测试方法，使外层循环的迭代参数取最小值，并为越界值或非法值增加一些额外测试。
  - ✓由内向外，对下一个循环进行测试，但保持所有外层循环为最小值，其他嵌套循环为“典型”值。
  - ✓继续进行下去，直到测试完所有循环。



- 串接循环

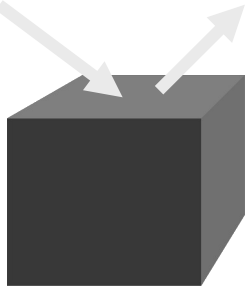
✓串接循环的各个循环彼此独立，可采用简单循环测试方法。

✓若前一个循环的计数器值是后一个循环的初始值，可采用嵌套循环测试方法。



二、黑盒测试技术

不考虑程序的内部结构与特性，只根据程序功能或程序的外部特性设计测试用例。



- 等价分类法
- 边值分析法
- 错误推测法
- 因果图法

二、黑盒测试技术

1. 等价类划分

- 基本思想：将被测程序所有可能的输入数据（有效、无效）划分为若干等价类，测试每个等价类的代表值，也就等于测试了该类的其它值。
- 分两步进行：
  - ✓划分等价类
  - ✓设计测试用例

（1）划分等价类

●首先从程序的功能说明中找出一个个输入条件，然后为每个条件划分合理等价类和不合理等价类，构造如下表格。

等价类表

输入条件	有效等价类	无效等价类
……	……	……
……	……	……
……	……	……

（1）划分等价类

●等价类划分规则：

① 如果规定了输入值的范围，如在编职工的年龄为 18-60 岁，则可划分出：

✓一个有效等价类： 18 岁 -60 岁之间

✓两个无效等价类：小于 18 岁，大于 60 岁

② 如果规定了输入数据的个数，如标识符由 2-6 个字符组成，则可划分出：

✓一个有效等价类： 2-6 之间

✓两个无效等价类：小于 2 ，大于 6

●等价类划分规则：

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，此外还有一个无效等价类（任一个不允许的输入值）。

④ 如果规定了输入数据的类型，则可以设置一个有效等价类和其它类型数据的若干无效等价类。

⑤ 如果规定了输入数据必须遵循的规则，可划分出一个符合规则的有效等价类，和若干个不满足各条规则的无效等价类（从各个不同角度违反规则）。

⑥ 如果程序的处理对象是表格，则应该考虑使用空表，以及含一项或多项的表的情况。

（2）设计测试用例

- 为每个等价类编号（唯一的编号）；
- 设计一个测试用例，使其尽可能多的覆盖尚未覆盖的多个有效等价类，重复这一步骤，直到所有有效等价类被覆盖为止；
- 设计一个新的测试用例，使其只覆盖一个尚未覆盖的无效等价类，重复这一步骤，直到所有无效等价类均被覆盖为止。

等价类划分举例：

- 用 PASCAL 语言编写的一个把数字串转换成整数的函数。运行程序的计算机字长 16 位，用二进制补码表示整数。该例被处理的数字串是右对齐的，即输入测试的数字串时，若少于六个字符左边补空格（“　123”）；若为负，负号在最高位左边一位（“-00123”）。

-32768 < 数字串 < 32767

- 划分等价类：

输入条件	有效等价类	无效等价类
6位字符串	(1) 最高位不是零 (2) 最高位是零 (3) 最高位前带负号	(4) 空串（全是空格） (5) 左部填充不是零也不是空格 (6) 最高位数字右部有空格 (7) 最高位数字右部有其它字符 (8) 负号与最高位之间有空格
数字串范围	(9) -32768～0 (10) 0 (11) 0～32767	(12) ＜-32768 (13) ＞32767

- 为有效等价类设计测试用例：

输入条件	有效等价类	测试数据	覆盖范围
6位字符串	(1)最高位不是零 (2)最高位是零 (3)最高位前带负号	“　　1” “000001” “-00001”	(1)、(11) (2)、(11) (3)、(9)
数字串范围	(9) -32768～0 (10) 0 (11) 0～ 32767	“000000”	(10)

- 为每个无效等价类至少设计一个测试用例：

无效等价类	测试数据	覆盖范围
(4) 空串（全是空格）	“　　”	(4)
(5)左部填充不是零也不是空格	“*****1”	(5)
(6) 最高位数字右部有空格	“1　2”	(6)
(7) 最高位数字右部有其它字符	“1**2”	(7)
(8) 负号与最高位之间有空格	“- 12”	(8)
(12) ＜-32768	“-47561”	(12)
(13) ＞32767	“132767”	(13)

二、黑盒测试技术

2. 边界值分析

- 基本思想：选取刚好等于、稍小于、稍大于等价类边界值的数据作为测试数据，而不是选取每个等价类内的典型值或任意值作为测试数据。
  - 如上例：　输入‘-32768’　　预期输出：　-32768
    输入‘32767’　　预期输出：　32767

二、黑盒测试技术

3. 错误推测

- 基本思想：列出程序中可能有的错误和容易发生错误的特殊情况，根据它们选择测试用例。
  - 根据：直觉、经验
  - 工具：常见错误清单、判定表等。

三、实用测试策略

黑盒 + 白盒

- 在任何情况下，都应使用边界值分析方法；
- 必要时用等价划分法补充测试用例；
- 必要时再用错误推测法补充测试用例；
- 对照组件的逻辑，检查已设计出的测试用例。

四、白盒测试与黑盒测试对比

	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
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
性质	是一种确认技术，回答“我们在构造一个正确的系统吗？”	是一种验证技术，回答“我们在正确地构造一个系统吗？”

第八节　软件调试

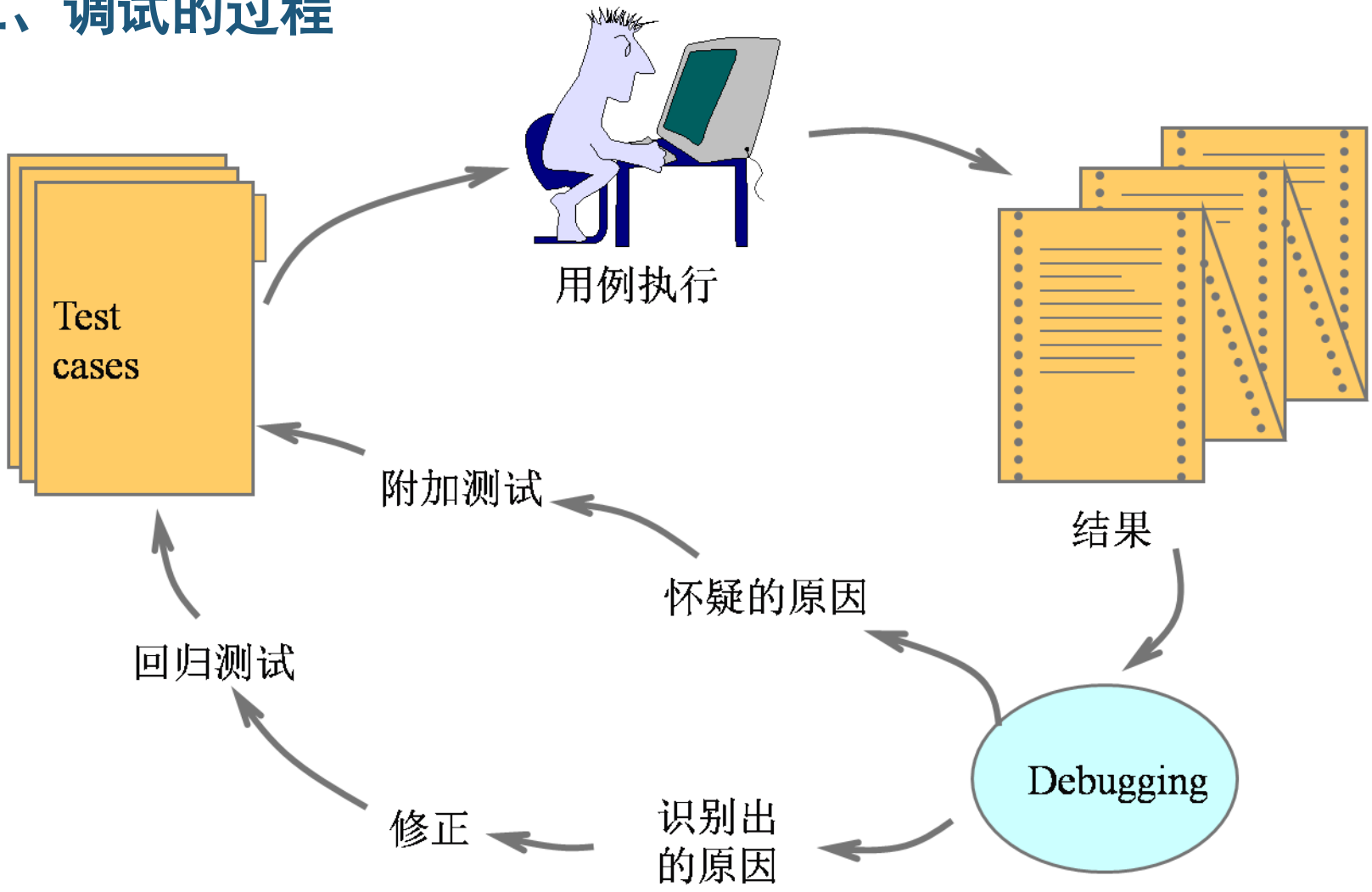
测试 —— 发现错误

调试 —— 改正错误

一、调试的原则

1. 注意错误的“群集现象”；
2. 不能只修改错误的征兆、表现。还应该修改错误的本质；
3. 注意在修改一个错误的同时，不要引入新的错误。

二、调试的过程



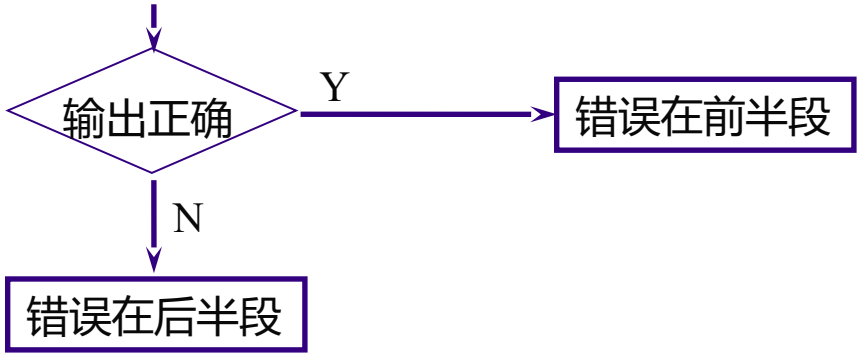
二、调试策略

调试过程的关键不是调试技术，而是用来推断错误原因的基本策略。主要有：

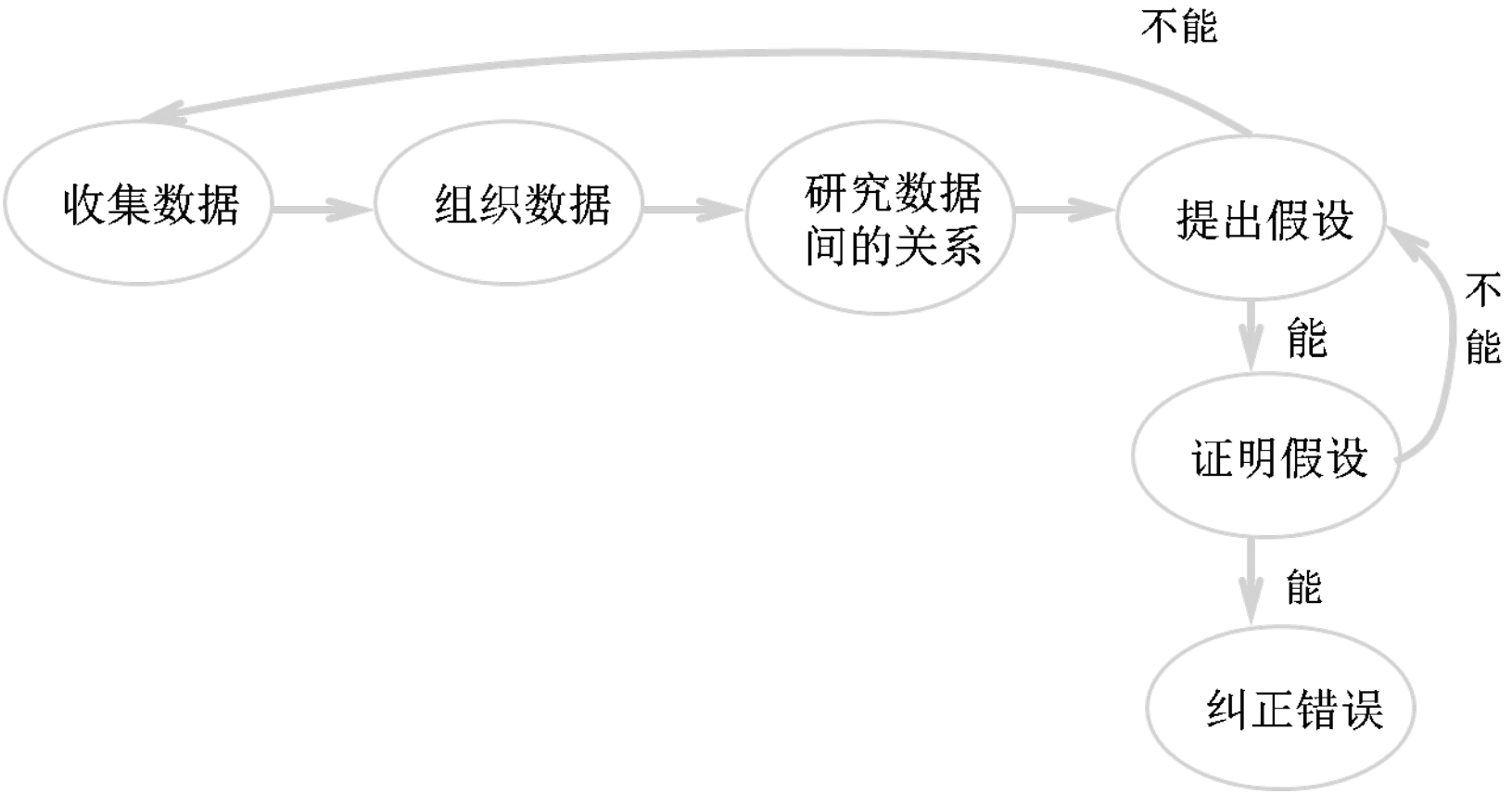
① 试探法，凭经验猜测。

② 回溯法：由症状（symptom）最先出现的地方，沿 control flow 向回检查。适用于小型程序。

③ 对分法：在关键点插入变量的正确值，则：

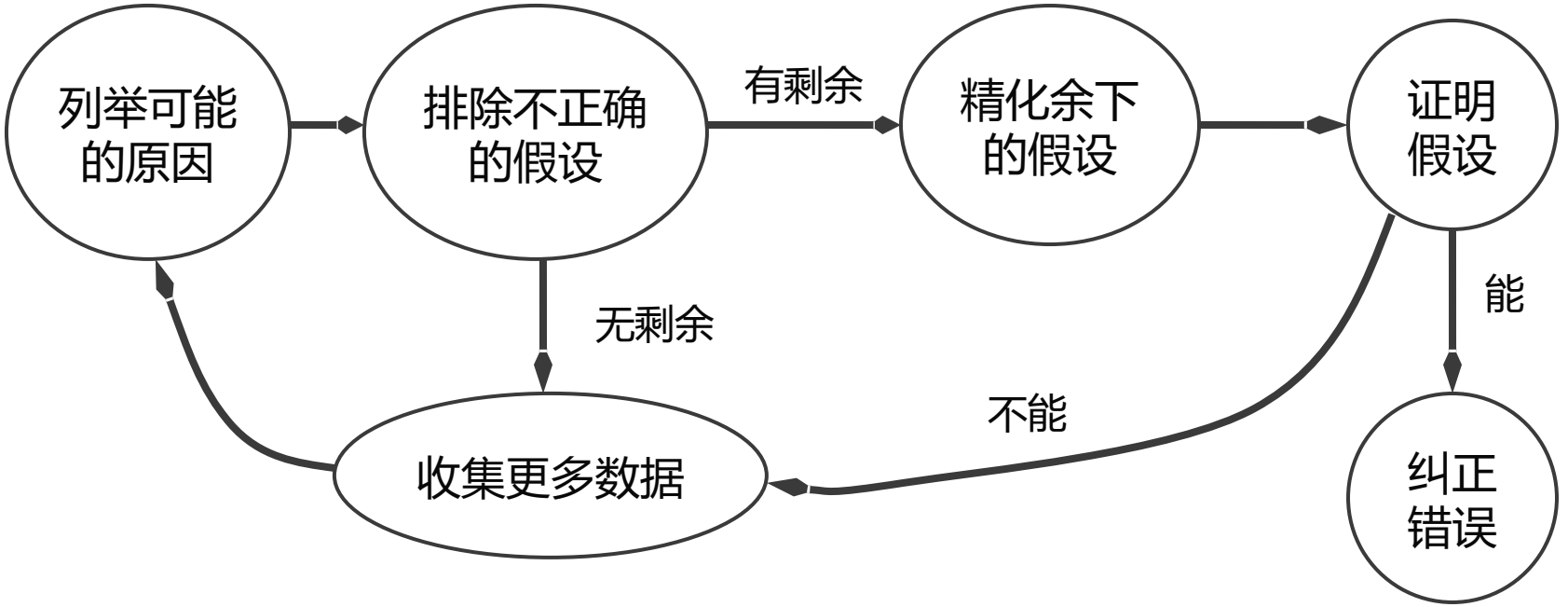


④ 归纳法：从错误症状中找出规律，推断根源。



⑤ 演绎法：普通 □ 特殊

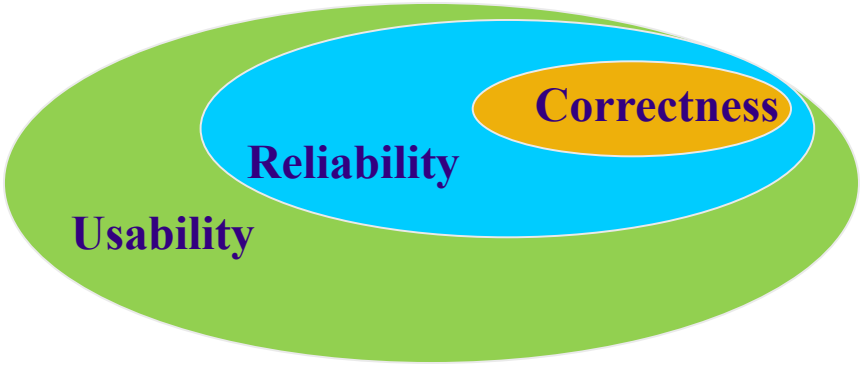
从假设中逐步排除、精化，从而导出错误根源。



第九节　软件可靠性

一、基本概念

- 可靠性（Reliability）：程序在给定的时间间隔内，按照说明书的规定，成功地运行的概率。
- 可用性（Usability）：程序在给定的时间点，按照说明书的规定，成功地运行的概率。
- 正确性（Correctness）：程序的功能正确。



MTTF　平均无故障时间　　MTTR　平均维修时间

设系统故障停机时间为 t_{d1}，t_{d2}，…；正常运行时间为 t_{u1}，t_{u2}，… ；则系统的“稳态可用性”为

$$\text{Availability} = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}}$$ (Shooman, 1983)

其中 MTTF = Mean Time To Failure = $\frac{1}{n}\sum_{i=1}^{n} t_{ui}$

MTTR = Mean Time To Repair = $\frac{1}{n}\sum_{i=1}^{n} t_{di}$

二、估算平均无故障时间

经验表明：

$$MTTF = \frac{1}{K(E_T/I_T - E_C(\tau)/I_T)}$$

其中：K 为经验常数（典型值约在 200 左右）；

E_T 为测试前故障总数；

I_T 为程序长度（机器指令总数）；

为测试（包括调试）时间；

E_C（　　）为时间从 0 至　　期间改正的错误数。

前提假设：

① E_T/I_T □ Constant （通常为 0.5~ 2% ）

② 调试中没有引入新故障（即 E_T 与时间　无关）

③ MTTF 与剩余故障成反比

换个角度看问题：

$$E_C(\tau) = E_T - \frac{I_T}{K \cdot MTTF}$$

意义：可根据对软件平稳运行时间的要求，估算需改正多少个错误后才能结束测试。

三、估算错误总数的方法

1. 植入故障法

人为植入 N_S 个故障，测后发现 n_s 个植入故障和 n 个原有故障，则设：

$$\frac{n_S}{N_S}=\frac{n}{\hat{N}} \Rightarrow \hat{N}=\frac{n}{n_S}N_S \approx E_T$$

2. 分别测试法

二人（组）分别独立测试同一程序，甲测得故障总数为 B_1，乙测得为 B_2，b_c 是发现相同的故障数，设以甲的测试结果为基准，则设：

$$\frac{b_c}{B_1}=\frac{B_2}{\hat{B}_0} \Rightarrow \hat{B}_0=\frac{B_2}{b_c}B_1 \approx E_T$$

一般多测几个 $\hat{B}_0$ 取平均。

举例

例：对一个包含 10,000 条机器指令的程序进行一个月集成测试后，总共改正了 15 个错误，此时 MTTF=10h，经过两个月测试后，总共改正了 25 个错误（第二个月改正了 10 个错误），MTTF=15h。 要求：

（1）根据上述数据确定 MTTF 与测试时间的函数关系，画出 MTTF 与测试时间 τ 的关系曲线。在画这条曲线时你做了什么假设？

（2）为使 MTTF=100h，必须进行多长时间的测试？当测试结束时总共改正了多少错误？还有多少错误潜藏在程序中？

解：（1）假设在 MTTF 和 τ 之间存在线性关系，即

$$MTTF = a + b\tau$$

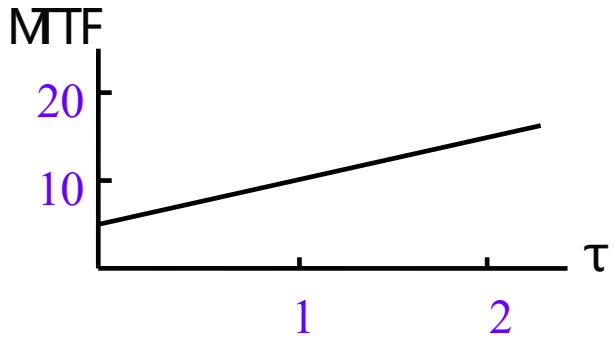
根据题意有：

$$\begin{cases} a + b = 10 \quad (\tau = 1) \\ a + 2b = 15 \quad (\tau = 2) \end{cases}$$

解得：　$a = 5$，　$b = 5$

因此，MTTF 与 τ 之间有下列关系为：　$MTTF = 5 + 5\tau$

根据上列方程式画出 MTTF 与 τ 之间的关系曲线如右图所示：



解：（2）为使 MTTF=100h，需要的测试时间

由　100　＝　5 + 5τ　　解得：　τ　＝ 19

即需要进行 19 个月的测试。

由：$$MTTF = \frac{I_T}{K\times(E_T - E_c)}$$ 有：$$\begin{cases} 10 = \dfrac{10000}{K\times(E_T - 15)} & ① \\ 15 = \dfrac{10000}{K\times(E_T - 25)} & ② \end{cases}$$

解得故障总数 E_T= 45 、 k≈33

由：$$100 = \frac{10000}{33\times(45 - E_c(19))}$$

解得：E_c = 42）

即改正了 42 个错误，还有 45-42=3 个潜藏的错误。


Thank
You