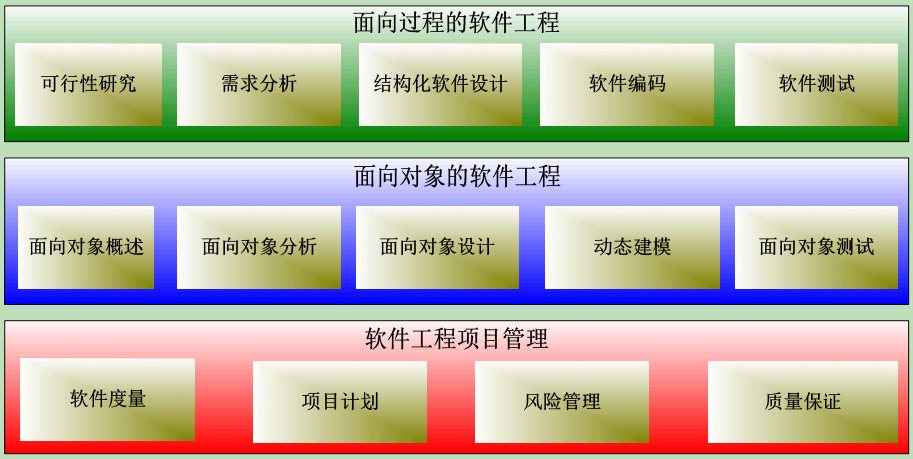
软件工程
第7讲 软件测试

贾西平
Email: jiaxp@126.com

课程主要内容



内容摘要

- 软件测试基础
- 白盒测试
- 黑盒测试
- 测试计划及分析报告
- 测试策略

3

内容摘要

- 软件测试基础
- 白盒测试
- 黑盒测试
- 测试计划及分析报告
- 测试策略

4

软件测试基础

- 软件测试的目的
- 软件测试的基本原则
- 白盒测试和黑盒测试

5

有关软件测试的错误观点

“软件测试是为了证明程序是正确的，即测试能发现程序中所有的错误”。

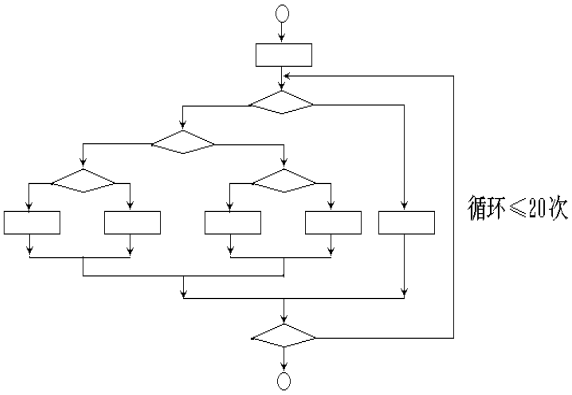
事实上这是不可能的。要通过测试发现程序中的所有错误，就要穷举所有可能的输入数据。

对于一个输入三个16位字长的整型数据的程序，输入数据的所有组合情况有$2^{48} \approx 3*10^{14}$，如果测试一个数据需1ms，则即使一年365天一天24小时不停地测试，也需要约1万年。

6

对一个具有多重选择和循环嵌套的程序，不同的路径数目可能是天文数字。

例如一个小程序的流程图，它包括了一个执行**20**次的循环，其循环体有五个分支。这个循环的不同执行路径数达**5^{20}条**，如果对每一条路径进行测试需要1毫秒，那么即使一年工作365 × 24小时，要想把所有路径测试完，大约需**3170年**。



测试只能证明程序中有错误，不能证明程序中没有错误！

7

软件测试的目的

- **软件测试目的**
  - **测试是一个为了发现错误而执行程序的过程**
  - **一个好的测试用例是指很可能找到迄今为至尚未发现的错误的测试用例**
  - **一个成功的测试是指揭示了迄今为至尚未发现的错误的测试**
- **测试的目标**

  以耗费最少时间与最小工作量找出软件系统中潜在的各种错误与缺陷。

8

软件测试任务

- 寻找Bug
- 避免软件开发过程中的缺陷
- 衡量软件的品质
- 关注用户的需求

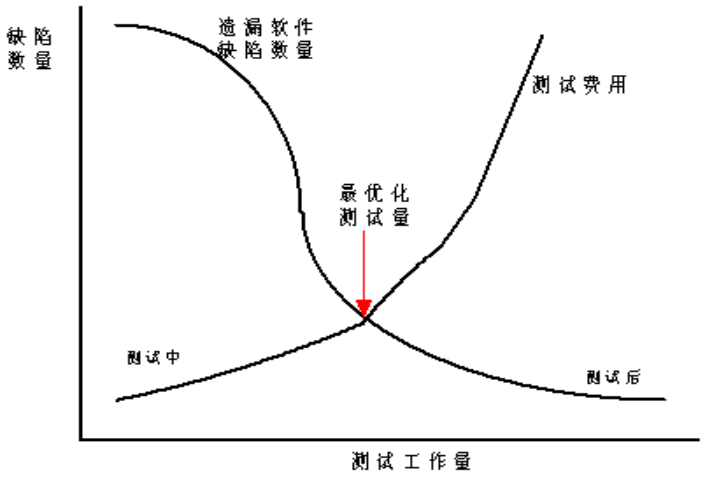
9

软件测试的基本原则

- 尽早地并不断地进行软件测试
- 程序员或程序设计机构应避免测试自己设计的程序
- 测试用例中不仅要有输入数据，还要有与之对应的预期结果
- 测试用例既要有合法的输入数据，还要有非法的输入数据
- 在对程序修改之后要进行**回归测试**
  - 回归测试：修改后， 重新测试修改前测试过的内容，防止修改引起错误
- 程序中尚未发现的错误的数量通常与该程序中已发现的错误的数量成正比
- 妥善保留测试文档（测试计划、全部测试用例、出错统计和最终分析报告），将其作为软件的组成部分之一，为维护提供方便
- 对每一个测试结果做全面检查
  - 避免有错误征兆的结果被漏掉
- 严格执行测试计划，排除测试的随意性

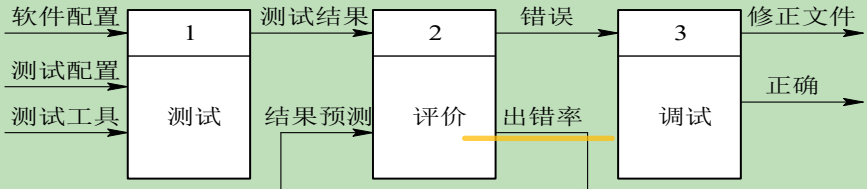
10

测试最佳平衡点



11

软件测试的过程（1）

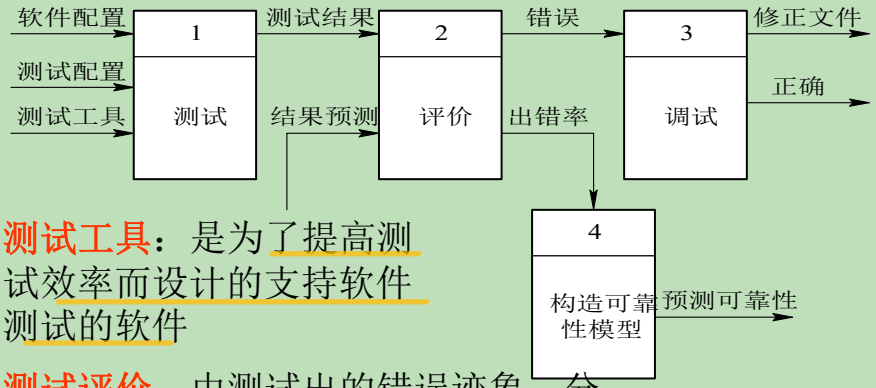


软件配置：指被测试软件的文件，如软件需求规格说明书、软件设计说明书和源程序清单等文档

测试配置：指测试方案、测试计划、测试用例、测试驱动程序等文档。

12

软件测试的过程（2）

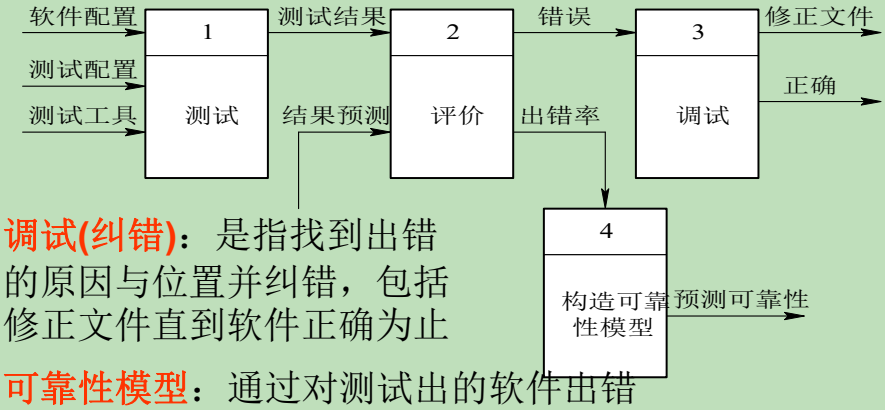


测试工具：是为了提高测试效率而设计的支持软件测试的软件

测试评价：由测试出的错误迹象，分析、找出错误的原因和位置，积累软件设计的经验。

13

软件测试的过程（3）



调试(纠错)：是指找到出错的原因与位置并纠错，包括修正文件直到软件正确为止

可靠性模型：通过对测试出的软件出错率的分析，建立模型，对软件可靠性进行预测，指导软件的设计与维护

14

软件测试技术

按照测试过程是否执行待测试软件分类：

- **静态分析技术**
  不执行被测试软件，通过分析软件的各种文档、源程序、流图、符号等发现错误
- **动态测试技术**
  执行被测试软件，由执行结果分析软件可能出现的错误。

15

软件测试方法分类

- 分析方法
  - 静态分析法
  - 白盒测试法
- 非分析法
  - 黑盒测试法

16

内容摘要

- 软件测试基础
- **白盒测试**
- 黑盒测试
- 测试计划与分析报告
- 测试策略

17

白盒测试

- **白盒测试（又称结构测试，逻辑驱动测试）**

  把测试对象看作一个透明的盒子，测试人员根据程序内部的逻辑结构及有关信息设计测试用例，检查程序中所有逻辑路径是否都按预定的要求正确地工作。

- **白盒测试主要用于对模块的测试，包括：**
  - **程序模块中的所有独立路径至少执行一次**
  - **对所有逻辑判定的取值（“真”与“假”）都至少测试一次**
  - **在上下边界及可操作范围内运行所有循环**
  - **测试内部数据结构的有效性等**

18

白盒测试

常用白盒测试方法：

- 语句覆盖
- 判定覆盖
- 条件覆盖
- 判定/条件覆盖
- 条件组合覆盖
- 路径覆盖
- 点覆盖
- 边覆盖

19

测试实例

例：对下列子程序进行测试

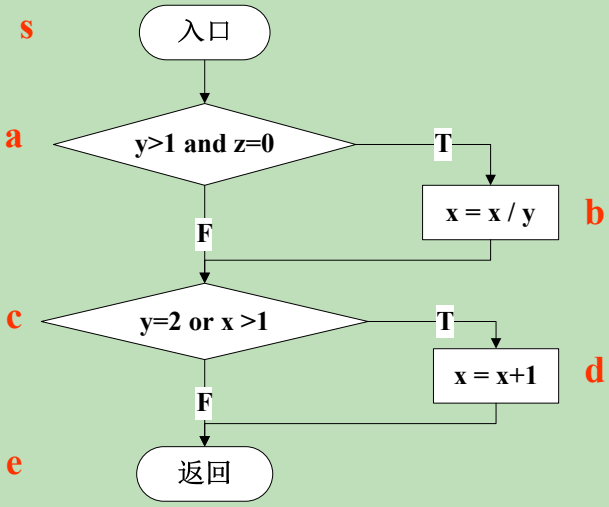
```
float example(float x, float y, float z){
   if (y>1 && z==0)
        x=x/y;
    if (y==2 || x>1)
        x=x+1;
   return x;
}
```

该段程序接受x、y、z的值，并将计算结果x的值返回给调用程序。

对应的流程图如下：

20

测试实例

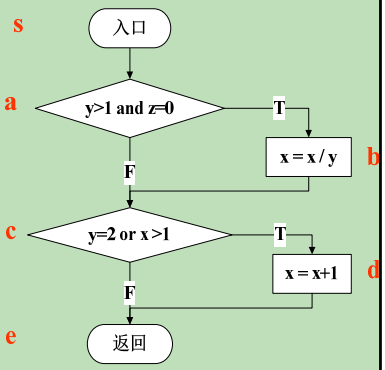


21

路径分析

该子程序有两个判定：
a: (y>1) and (z=0)
c: (y=2) or (x>1)
判定a中有两个判定条件：
y>1; z=0
判定c中有两个判定条件：
y=2 ; x>1
根据程序的执行流程不同，判定c中的“x>1”的含义也不同

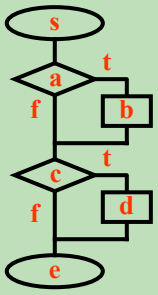
- **当判定a为“真”时，“x>1”实际是“x/y>1”，即“x>y”；**
- **当判定a为“假”时，“x>1”仍是“x>1”**



22

路径分析

该程序段有四条可执行路径：
路径1 sabcde ，其执行条件（L1）
是a为"T"且c为"T"
L1= {(y>1) and (z=0)} and
{(y=2) or (x/y>1)}
= (y>1) and (z=0) and (y=2) or
(y>1) and (z=0) and (x> y)
= **(y=2) and (z=0) or**
(y>1) and (z=0) and (x> y)

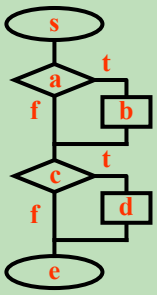


a: (y>1) and (z=0)
c: (y=2) or (x>1)

23

路径分析

路径2 sace ，其执行条件（L2）是
a为"F"且c为"F")
L2 = not{(y>1) and (z=0)} and
not{(y=2) or (x>1)}
= { not (y>1) or not (z=0) } and
{ not (y=2) and not (x>1) }
= not (y>1) and not (y=2) and not (x>1)
or
not (z=0) and not (y=2) and not (x>1)
=(y≤1) and (x ≤ 1) or
(z ≠ 0) and (y ≠ 2) and (x ≤ 1)



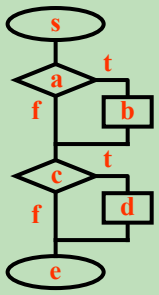
a: (y>1) and (z=0)
c: (y=2) or (x>1)

24

路径分析

路径3 sacde ，其执行条件（L3）是a为”F”且c为”T”)

L3 = not {(y>1) and (z=0)} and
{(y=2) or (x>1)}
= { not (y>1) or not (z=0)} and
{(y=2) or (x>1)}
= not (y>1) and (y=2) or not (y>1) and (x>1)
or not (z=0) and (y=2) or
not (z=0) and (x>1)
= **(y ≤ 1) and (y=2)** or (y ≤ 1) and (x>1) or
(z ≠ 0) and (y=2) or (z ≠ 0) and (x>1)
= (y ≤ 1) and (x>1) or (z ≠ 0) and (y=2) or (z ≠ 0) and (x>1)



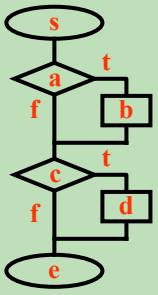
a: (y>1) and (z=0)
c: (y=2) or (x>1)

25

路径分析

路径4 sabce ，其执行条件（L4）是a为”T”且c为”F”)

L4 = {(y>1) and (z=0)} and
not {(y=2) or (x/y>1)}
= (y>1) and (z=0) and not (y=2)
and not (x> y)
=(y>1) and (z=0) and (y ≠ 2) and (x ≤ y)



a: (y>1) and (z=0)
c: (y=2) or (x>1)

26

语句覆盖

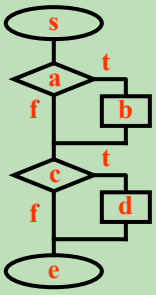
语句覆盖是指选择足够的测试用例，使得被测程序的每个可执行语句都至少执行一次。

欲使每个语句都执行一次，只需执行路径L1（sabcde）即可。

L1= (y=2) and (z=0) or
(y>1) and (z=0) and (x> y)

测试用例如下：

测试数据	预期结果
x=4,y=2,z=0	x=3



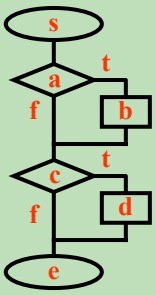
a: (y>1) and (z=0)
c: (y=2) or (x>1)

27

判定覆盖

判定覆盖（也称分支覆盖）：选择足够的测试用例，使得被测程序的每个判定框的真假分支都至少执行一次。

欲使每个分支都执行一次，只需执行路径L3（sacde, a为”F”且c为”T”）和L4（sabce, a为”T”且c为”F”）。
或者，执行路径L1（sabcde, a为”T”且c为”T”）和L2（sace, a为”F”且c为”F”）.

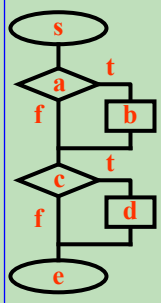


a: (y>1) and (z=0)
c: (y=2) or (x>1)

28

判定覆盖

L3（sacde, a为"f"且c为"t"）：
(y ≤ 1) and (x>1) or (z ≠ 0) and (y=2) or (z ≠ 0) and (x>1)
L4（sabce, a为"t"且c为"f"）：
(y>1) and (z=0) and (y ≠ 2) and (x ≤ y)



a: (y>1) and (z=0)
c: (y=2) or (x>1)

测试数据	预期结果	路径	a	c
x=1,y=2,z=1	x=2	sacde	f	t
x=3,y=3,z=0	x=1	sabce	t	f

29

判定覆盖

- 判定覆盖将每个判定的所有可能结果都至少执行一次
- 满足判定覆盖标准的测试用例也一定满足语句覆盖标准。

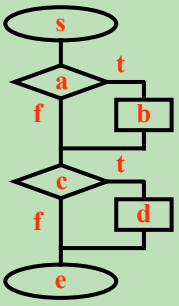
30

条件覆盖

条件覆盖: 选择足够的测试用例，使得被测程序的每个判定中的每个条件的所有可能取值都至少执行一次。

判定a中各种条件的所有可能结果：y>1，y ≤ 1 ，z=0， z ≠ 0。

判定c中各种条件的所有可能结果：y=2，y ≠ 2 ，x>1（或x>y），x ≤ 1 (或x≤y)。



a: (y>1) and (z=0)

c: (y=2) or (x>1)

测试数据	预期结果	路径	覆盖的条件
x=1,y=2,z=0	x=1.5	sabcde	y>1， z=0，y=2，x ≤ y
x=2,y=1,z=1	x=3	sacde	y ≤ 1，z ≠ 0，y ≠ 2 ，x>1

31

注意

- 条件覆盖不一定包含判定覆盖
- 判定覆盖也不一定包含条件覆盖

32

判定/条件覆盖

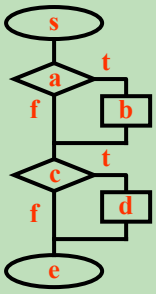
判定/条件覆盖：选择足够的测试用例，使得判断中**每个条件**的**所有可能取值**至少执行一次，同时**每个判断**的**所有可能判断结果（分支）**至少执行一次。

- 满足判定/条件覆盖标准的测试用例一定也满足判定覆盖、条件覆盖、语句覆盖标准。

33

判定/条件覆盖

a: (y>1) and (z=0)
c: (y=2) or (x>1)



测试数据	预期结果	路径	a	c	覆盖的条件
x=4,y=2,z=0	x=3	sabcde	t	t	y>1，z=0，y=2，x > y
x=1,y=1,z=1	x=1	sace	f	f	y ≤ 1，z ≠ 0，y ≠ 2 ，x ≤1

34

条件组合覆盖

条件组合覆盖：选择足够的测试用例，使得**每个判断的所有可能的条件取值组合**至少执行一次。

判定a中条件结果的所有可能组合：

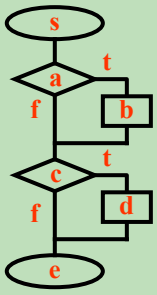
①y>1，z=0；　② y>1，z ≠ 0;

③y ≤ 1，z=0；④ y ≤ 1，z ≠ 0

判定c中条件结果的所有可能组合：

⑤ y=2，x>1;　⑥ y=2，x ≤ 1；

⑦ y ≠ 2，x>1;　⑧ y ≠ 2，x ≤ 1



a: (y>1) and (z=0)

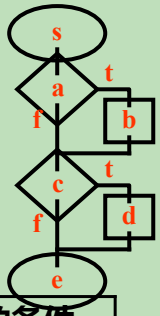
c: (y=2) or (x>1)

35

条件组合覆盖

a: (y>1) and (z=0)

c: (y=2) or (x>1)



测试数据	预期结果	路径	a	c	覆盖的条件
x=4,y=2,z=0	x=3	sabcde	t	t	①y>1，z=0 ⑤y=2，x>y
x=1,y=2,z=1	x=2	sacde	f	t	②y>1，z ≠ 0 ⑥y=2，x ≤ 1
x=2,y=1,z=0	x=3	sacde	f	t	③y ≤ 1，z=0 ⑦y ≠ 2，x>1
x=1,y=1,z=1	x=1	sace	f	f	④y ≤ 1，z ≠ 0 ⑧y ≠ 2，x≤1

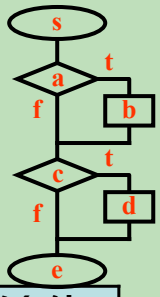
36

条件组合覆盖

不能保证程序中所有可能的路径都被覆盖。

本例中，满足条件组合覆盖标准的测试用例就没有经过sabce路径。

a: (y>1) and (z=0)
c: (y=2) or (x>1)



测试数据	预期结果	路径	a	c	覆盖的条件
x=4,y=2,z=0	x=3	sabcde	t	t	①y>1，z=0 ⑤y=2，x>y
x=1,y=2,z=1	x=2	sacde	f	t	②y>1，z ≠ 0 ⑥y=2，x ≤ 1
x=2,y=1,z=0	x=3	sacde	f	t	③y ≤ 1，z=0 ⑦y ≠ 2，x>1
x=1,y=1,z=1	x=1	sace	f	f	④y ≤ 1，z ≠ 0 ⑧y ≠ 2，x≤1

37

路径覆盖

路径覆盖：选择足够的测试用例，使得被测程序的每条可能路径都至少执行一次（如果程序中包含环路，则要求每条环路至少经过一次）。

38

路径覆盖

本例中所有可能执行的路径有：

L1 (sabcde ， a为”t”且c为”t”)

= (y=2) and (z=0) or

(y>1) and (z=0) and (x> y)

L2 (sace ， a为”f”且c为”f”)

=(y≤1) and (x ≤ 1) or

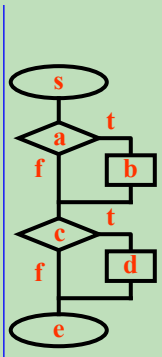
(z ≠ 0) and (y ≠ 2) and (x ≤ 1)

L3 (sacde ， a为”f”且c为”t”)

= (y ≤ 1) and (x>1) or (z ≠ 0) and (y=2) or (z ≠ 0) and (x>1)

L4 (sabce ， a为”t”且c为”f”)

=(y>1) and (z=0) and (y ≠ 2) and (x ≤ y)

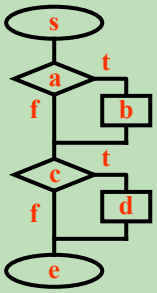


a: (y>1) and (z=0)
c: (y=2) or (x>1)

39

路径覆盖

测试数据	预期结果	路径	a	c
x=4,y=2,z=0	x=3	sabcde	t	t
x=3,y=3,z=0	x=1	sabce	t	f
x=2,y=1,z=0	x=3	sacde	f	t
x=1,y=1,z=1	x=1	sace	f	f



a: (y>1) and (z=0)
c: (y=2) or (x>1)

40

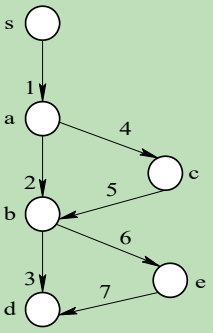
路径覆盖

- 路径覆盖实际上考虑了程序中各种判断结果的所有可能组合
- 是一种相对较强的逻辑覆盖标准
- 没有检验表达式中条件的各种组合情况
- **不能替代**条件覆盖和条件组合覆盖标准

41

点覆盖

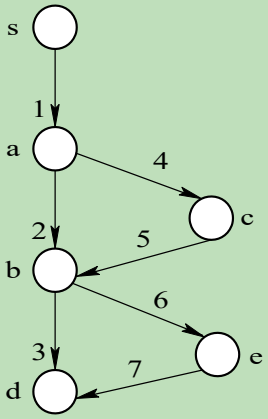
- **点覆盖：**设计足够的测试用例，使程序执行时至少经过程序图中每个节点一次。
- **程序图**：连通的有向图，图中每个节点相当于程序流程图中的一框(一个或多个语句)
- 点覆盖相当于语句覆盖。



42

边覆盖

- **边覆盖：**设计足够的测试数据，使得程序执行路径至少经过程序图中每一个边一次。
- 如右图，为了使程序执行路径经过程序图的边覆盖(1，2，3，4，5，6，7)，至少需要两组测试数据(分别执行路径1-2-3和1-4-5-6-7，或分别执行路径1-4-5-3和1-2-6-7)。



43

内容摘要

- 软件测试基础
- 白盒测试
- **黑盒测试**
- 测试计划与分析报告
- 测试策略

44

黑盒测试

- **黑盒测试：**又称功能测试，把测试对象看做一个黑盒子，不考虑程序内部的逻辑结构和处理过程，只按照软件的需求规约，检查程序的功能是否符合需求规约的要求。
- 试图发现以下类型错误：
  - 不正确或遗漏的功能
  - 接口错误，如输入/输出参数的个数、类型等
  - 数据结构错误或外部信息（如外部数据库）访问错误
  - 性能错误
  - 初始化和终止错误

45

黑盒测试方法

- 主要的黑盒测试方法有：
  - 等价类划分
  - 边界值分析
  - 因果图
  - 错误推测

46

等价类划分

- 选择**少量有代表性**的输入数据，揭露尽可能多的程序错误
- 等价类划分方法将所有可能的输入数据划分成若干个等价类，然后在每个等价类中选取一个代表性的数据作为测试用例
  - 等价类是指输入数据集合的某个子集，该子集中的每个输入数据对揭露软件中的错误都是等效的。
  - 等价类的划分主要依靠测试人员的经验。

47

等价类划分

- **等价类划分方法: 把输入数据分为有效输入数据和无效输入数据**
- **有效输入数据指符合规格说明要求的合理的输入数据，主要用来检验程序是否实现了规格说明中的功能**
- **无效输入数据指不符合规格说明要求的不合理或非法的输入数据，主要用来检验程序是否做了规格说明以外的事**
- **在确定输入数据等价类时，常常还要分析输出数据的等价类，以便根据输出数据等价类导出输入数据等价类。**

48

等价类划分设计测试用例的步骤

- **确定等价类**

根据软件的规格说明，对每一个输入条件（通常是规格说明中的一句话或一个短语）确定若干个有效等价类和若干个无效等价类。

可使用如下表格

输入条件	有效等价类	无效等价类

49

确定等价类的规则

(1) 如果输入条件规定了取值范围，则可以确定一个有效等价类（输入值在此范围内）和两个无效等价类（输入值小于最小值及大于最大值）

例如，规定输入的考试成绩在0..100之间，则有效等价类是“0 ≤ 成绩 ≤ 100”，无效等价类是“成绩 < 0”和“成绩 > 100”。

50

确定等价类的规则

(2) 如果输入条件规定了值的个数，则可以确定一个有效等价类（输入值的个数等于规定的个数）和两个无效等价类（输入值的个数小于规定的个数和大于规定的个数）

例如，规定输入构成三角形的3条边，则有效等价类是“输入边数 = 3”，无效等价类是“输入边数 < 3”和“输入边数 > 3”。

51

确定等价类的规则

(3) 如果输入条件规定了输入值的集合（即离散值），而且程序对不同的输入值做不同的处理，那么每个允许的值都确定为一个有效等价类，另外还有一个无效等价类（任意一个不允许的值）。

例如，规定输入的考试成绩为优、良、中、及格、不及格，则可确定5个有效等价类和一个无效等价类。

52

确定等价类的规则

(4) 如果输入条件规定了输入值必须遵循的规则，那么可确定一个有效等价类（符合此规则）和一个无效等价类（不符合此规则）。

例如，在Pascal语言中对变量标识符规定为“以字母开头的……串”。那么有效等价类是“以字母开头的串”，而无效等价类是“不以字母开头的串”。

53

确定等价类的规则

(5) 如果认为程序将按不同的方式来处理某个等价类中的各种测试用例，则应将这个等价类再分成几个更小的等价类。

(6) 如果输入条件是一个布尔量，则可以确定一个有效等价类和一个无效等价类。

(7) 如果输入条件规定输入数据是整型，那么可以确定三个有效等价类（正整数、零、负整数）和一个无效等价类（非整数）。

54

设计测试用例

- 在确定了等价类之后，建立等价类表，列出所有划分出的等价类。

输入条件	有效等价类	无效等价类

55

设计测试用例

利用等价类设计测试用例的步骤：

(1)为每个有效等价类和无效等价类编号

(2)设计一个新的测试用例，使其尽可能多地覆盖尚未被覆盖的有效等价类，重复这一步，直到所有的有效等价类都被覆盖为止；

(3) 为每个无效等价类设计一个新的测试用例

56

例：等价类划分法设计测试用例

- 例：
  某编译程序的规格说明中关于标识符的规定如下：
  - 标识符是由字母开头，后跟字母或数字的任意组合构成；
  - 标识符的字符数为1～8个；
  - 标识符必须先说明后使用；
  - 一个说明语句中至少有一个标识符；
  - 保留字不能用作变量标识符。

57

例：等价类划分法设计测试用例

用等价类划分方法，建立输入等价类表:

输入条件	有效等价类	无效等价类
第一个字符	字母(1)	数字(2) 非字母数字字符(3)
后跟的字符	字母(4) 数字(5)	非字母数字字符(6) 保留字(7)
字符数	1~8个(8)	0个(9) ＞8个(10)
标识符的使用	先说明后使用(11)	未说明已使用(12)
标识符个数	≥1个(13)	0个(14)

58

- 下面选取9个测试用例，它们覆盖了所有的等价类。

输入数据	预期结果	覆盖的等价类
VAR P3t2：REAL； BEGIN P3t2：=3．1； …… END；	正确标识符	(1)，(4)，(5)，(8)，(11)，(13)
VAR 3P：REAL；	报错：不正确标识符	(2)
VAR !X：REAL；	报错：不正确标识符	(3)
VAR T#：CHAR；	报错：不正确标识符	(6)

59

输入数据	预期结果	覆盖的等价类
VAR GOTO：INTEGER；	报错：保留字作标识符	(7)
VAR X，：REAL；	报错：标识符长度为0	(9)
VAR T12345678：REAL；	报错：标识符字符超长	(10)
VAR PAR：REAL； BEGIN …… PAP：=3．14 …… END；	报错：未说明已使用	(12)
VAR ：REAL；	报错：标识符个数为0	(14)

60

边界值分析

- 大量的错误是发生在输入或输出范围的边界上
- 针对各种边界情况设计测试用例，其揭露程序中错误的可能性较大。

61

边界值分析

- **边界是指相对于输入等价类和输出等价类而言，直接在其边界上、或稍高于其边界值、或稍低于其边界值的一些特定情况。**
- **边界值分析专门挑选那些位于边界附近的值（即正好等于、或刚刚大于、或刚刚小于边界的值）作为测试用例。**

62

边界值分析法选择测试用例原则

1．如果输入条件规定了值的范围，则选择刚刚达到这个范围的边界的值以及刚刚超出这个范围的边界的值作为测试输入数据。

例如，规定输入的考试成绩在0～100之间，则取0，100，－1，101作为测试输入数据。

2．如果输入条件规定了值的个数，则分别选择最大个数、最小个数、比最大个数多1、比最小个数少1的数据作为测试输入数据。

例如，规定一个运动员的参赛项目至少1项，最多3项，那么，可选择参赛项目分别是1项、3项、0项、4项的测试输入数据。

63

边界值分析法选择测试用例原则

3．对每个输出条件使用第1条。

例如，输出的金额值大于等于0且小于10^4，则选择使得输出金额分别为0、9999、－1、10000的输入数据作为测试数据。

4．对每个输出条件使用第2条。

例如，规定输出的一张发票上，至少有1行内容，至多有5行内容，则选择使得输出发票分别有1行、5行、0行、6行内容的输入数据作为测试数据。

5．如果程序的输入或输出是个有序集合（如顺序文件、线性表），应把注意力集中在有序集的第1个和最后一个元素上。

64

边界值分析法选择测试用例原则

6．如果程序中定义的内部数据结构有预定义的边界（例如，数组的上界和下界、栈的大小），则应选择正好达到该数据结构边界以及刚好超出该数据结构边界的输入数据作为测试数据。

例如，程序中数组A的下界是10，上界是20，则可选择使得A的下标为10、20、9、21的输入数据作为测试数据。

7．分析规格说明，找出其他可能的边界条件

65

因果图法

- 在等价类划分方法和边界值方法中未考虑输入条件的各种组合
- 因果图方法**主要检查各种输入条件的组合。**

66

因果图方法的特点

- 考虑输入条件的组合关系；
- 考虑输出条件对输入条件的依赖关系，即因果关系；
- 测试用例发现错误的效率高；
- 能检查出功能说明中的某些不一致或遗漏。

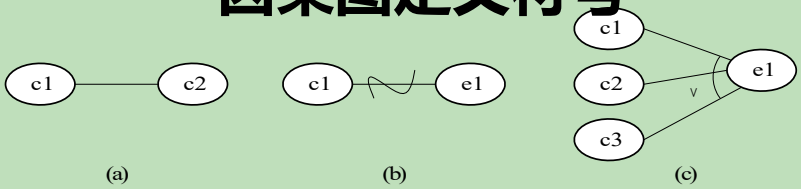
67

因果图的测试用例设计步骤

(1) 分析规格说明中的输入作为因，输出作为果。
(2) 依据因果的处理语义画出因果图。
(3) 标出因果图的约束条件。
(4) 将因果图转换为因果图所对应的判定表。
(5) 根据判定表设计测试用例。

68

因果图定义符号



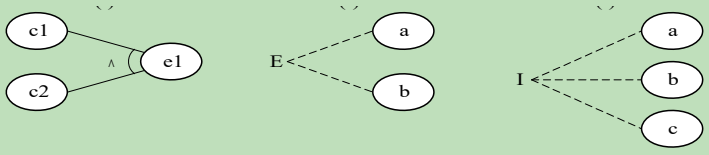
(a) **恒等**：表示原因与结果之间是一对一的对应关系。若原因出现，则结果出现。若原因不出现，则结果也不出现。

(b) **非**：表示原因与结果之间的一种否定关系。若原因出现，则结果不出现。若原因不出现，则结果出现。

(c) **或(∨)**：表示若几个原因中有一个出现，则结果出现，只有当这几个原因都不出现时，结果才不出现。

69

因果图定义符号



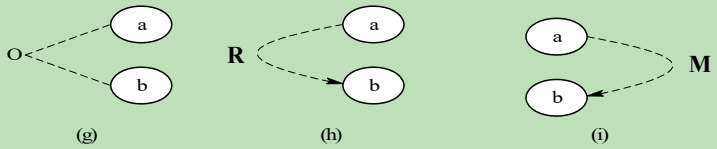
(d) 与(∧)：表示若几个原因都出现，结果才出现。若几个原因中有一个不出现，结果就不出现。

(e) 异约束（互斥）：表示a，b两个原因不会同时成立，两个中最多有一个可能成立。

(f) 或约束（包含）：表示a，b，c三个原因中至少有一个必须成立。

70

因果图定义符号



(g) (h) (i)

(g) 惟一约束：表示a和b原因当中必须有一个，且仅有一个成立。

(h) 要求约束：表示当a出现时，b必须也出现，不可能a出现，b不出现。

(i) 强制约束：它表示当a是1时，b必须是0，而当a为0时，b的值不定。

71

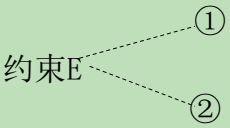
例：用因果图设计测试用例

某规格说明：“ 第一列字符必须是A或者B，第二列字符必须是一个数字，第一、二两列都满足时修改文件，第一列不正确时给出信息L，第二列不正确时给出信息M。”

（1）分析规格说明并编号。

因：第一列字符是A ①

第一列字符是B ②



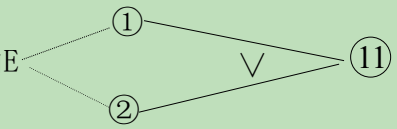
只有一个为1，不能同时为1

72

例：用因果图设计测试用例

第二列字符是数字 ③

果：一列正确E



⑪ = ①∨②

修改文件 ㉑ = ⑪∧③ 即 (①∨②)∧③

给出L信息 ㉒ = $\overline{⑪}$ 即 $\overline{①∨②}$

给出M信息 ㉓ = $\overline{③}$

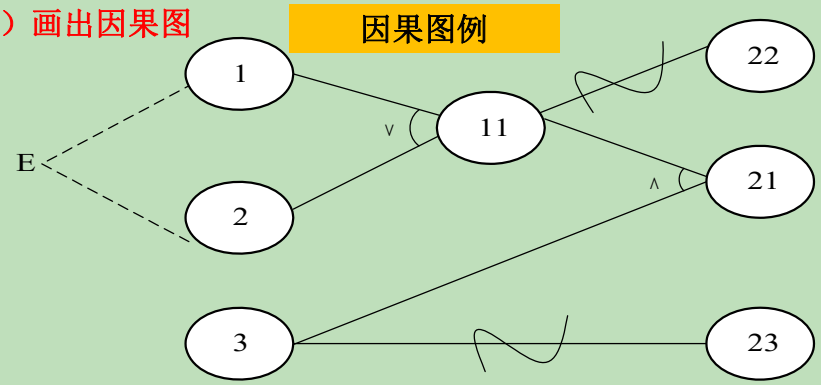
“第一列字符必须是A或者B，第二列字符必须是一个数字，第一、二两列都满足时修改文件，第一列不正确时给出信息L，第二列不正确时给出信息M。”

73

例：用因果图设计测试用例

（2）画出因果图

因果图例



修改文件 ㉑ = ⑪∧③ 即 (①∨②)∧③

给出L信息 ㉒ = $\overline{⑪}$ 即 $\overline{①∨②}$

给出M信息 ㉓ = $\overline{③}$

74

例：用因果图设计测试用例

(3) **将因果图转换为判定表**(见下页表)

- 遇到E约束记为X；
- 条件和输出结果编号成立时记为1，否则为0；
- 每一列视为测试规则

75

判 定 表

组合条件		1	2	3	4	5	6	7	8
条件原因	(1)	1	1	1	1	0	0	0	0
	(2)	1	1	0	0	1	1	0	0
	(3)	1	0	1	0	1	0	1	0
	(11)	X	X	1	1	1	1	0	0
动作结果	(21)	X	X	1	0	1	0	0	0
	(22)	X	X	0	0	0	0	1	1
	(23)	X	X	0	1	0	1	0	1

76

例：用因果图设计测试用例

(4) **由判定表的3～8列编写测试用例如下：**

根据3列　输入：A3，A8　　**输出**：**修改文件**

根据5列　输入：B4，B5　　**输出**：**修改文件**

根据4列　输入：AM，A？　**给出信息**M

根据6列　输入：BB，BC　　**给出信息**M

根据7列　输入：M1，X6　　**给出信息**L

根据8列　输入：XY，MN　**给出信息**M**与**L

77

举例（略）

- 问题描述：
  - 有一个处理单价为5角钱的饮料的自动售货机软件测试用例的设计。
  - 其规格说明如下：若投入5角钱或1元钱的硬币，押下【橙汁】或【啤酒】的按钮，则相应的饮料就送出来。若售货机没有零钱找，则一个显示【零钱找完】的红灯亮，这时在投入1元硬币并押下按钮后，饮料不送出来而且1元硬币也退出来；若有零钱找，则显示【零钱找完】的红灯灭，在送出饮料的同时退还5角硬币。

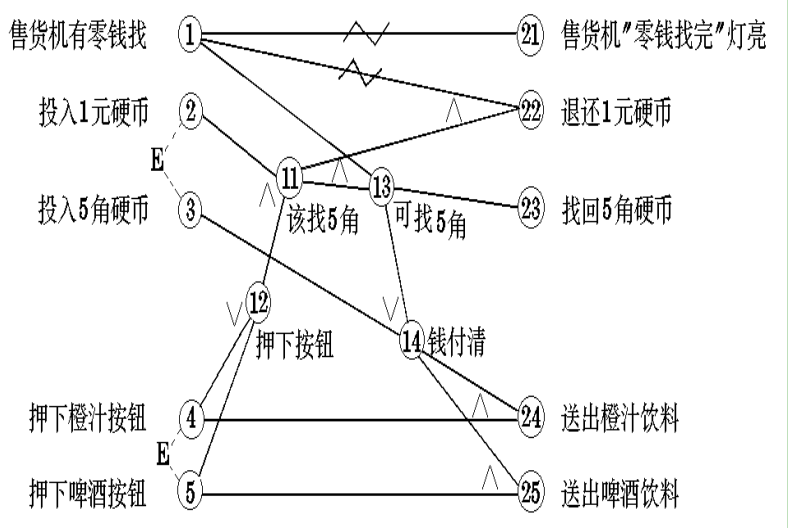
78

分析列出原因和结果

- 原因：
  1. 售货机有零钱找
  2. 投入1元硬币
  3. 投入5角硬币
  4. 押下橙汁按钮
  5. 押下啤酒按钮
- 建立中间结点，表示处理中间状态：
  11. 投入1元硬币且押下饮料按钮
  12. 押下【橙汁】或【啤酒】的按钮
  12. 应当找5角零钱并且售货机有零钱找
  12. 钱已付清
- 结果：
  21. 售货机【零钱找完】灯亮
  22. 退还1元硬币
  23. 退还5角硬币
  24. 送出橙汁饮料
  25. 送出啤酒饮料

79

画出因果图



80

加上约束条件

- 由于 2 与 3 ，4 与 5 不能同时发生，分别加上约束条件E
- 其他或、与关系

81

转换成判定表

序号		1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1	2
条件	①	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	②	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
	③	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
	④	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
	⑤	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
中间结果	⑪						1	1	0		0	0	0		0	0	0						1	1	0		0	0	0		0	0	0
	⑫						1	1	0		1	1	0		1	1	0						1	1	0		1	1	0		1	1	0
	⑬						1	1	0		0	0	0		0	0	0						0	0	0		0	0	0		0	0	0
	⑭						1	1	0		1	1	1		0	0	0						0	0	0		1	1	1		0	0	0
结果	㉑						0	0	0		0	0	0		0	0	0						1	1	1		1	1	1		1	1	1
	㉒						0	0	0		0	0	0		0	0	0						1	1	0		0	0	0		0	0	0
	㉓						1	1	0		0	0	0		0	0	0						0	0	0		0	0	0		0	0	0
	㉔						1	0	0		1	0	0		0	0	0						0	0	0		1	0	0		0	0	0
	㉕						0	1	0		0	1	0		0	0	0						0	0	0		0	1	0		0	0	0
测试用例							Y	Y	Y		Y	Y	Y		Y	Y							Y	Y	Y		Y	Y	Y		Y	Y	

82

错误推测法

- **思路：**列举出程序中所有可能的错误和容易发生错误的特殊情况，根据这些猜测设计测试用例。
- 主要依靠直觉和经验，没有机械的执行步骤。

83

错误推测法

- **例如，测试一个排序子程序，可考虑如下情况：**
  - **输入表为空；**
  - **输入表只有一个元素；**
  - **输入表的所有元素都相同；**
  - **输入表已排序。**

84

内容摘要

- 软件测试基础
- 白盒测试
- 黑盒测试
- **测试计划与分析报告**
- 测试策略

85

软件测试计划

1. 引言
- 1.1 编写目的
- 1.2 背景
- 1.3 定义
- 1.4 参考资料

2. 计划
- 2.1 软件说明
- 2.2 测试内容
- 2.3 测试1(标识符)
  - 2.3.1 进度安排
  - 2.3.2 条件
    - a. 设备
    - b. 软件
    - c. 人员
  - 2.3.3 测试资料
    - a. 有关本项任务的文件
    - b. 被测试程序及其所在的媒体
    - c. 测试的输入和输出举例
    - d. 有关控制此项测试的方法、过程的图表
  - 2.3.4 测试培训
- 2.4 测试2(标识符)

86

软件测试计划

3. 测试设计说明

3.1 测试1(标识符)

3.1 .1 控制

3.1.2 输入

3.1.3 输出

3.2 测试2(标识符)

……

4. 评价准则

4.1 范围

4.2 数据整理

4.3 尺寸

87

测试分析报告

1. 引言

1.1 编写目的

1.2 背景

1.3 定义

1.4 参考资料

2. 测试概要

3. 测试结果及发现

3.1 测试1(标识符)

3.2 测试2(标识符)

4. 对软件功能的结论

4.1 功能1(标识符)

4.1.1 能力

4.1.2 限制

4.2 功能2(标识符)

5. 分析摘要

5.1 能力

5.2 缺限和限制

5.3 建议

a. 各项修改可采用的修改方法程度

b. 各项修改的紧迫程度

c. 各项修改预定的工作量

d. 各项修改的负责人

5.4 评价

6. 测试资源消耗

88

内容摘要

- 软件测试基础
- 白盒测试
- 黑盒测试
- 测试计划与分析报告
- **测试策略**

89

测试策略

- **测试策略**:综合使用不同测试方法，以满足不同测试需要。
- 可用以下策略结合不同测试方法：
  (1) 任何情况下都应使用**边界值**分析法。
  (2) 必要时用**等价类划分法**补充测试方案。
  (3) 必要时用**错误推测法**补充测试方案。
  (4) 如果在程序的功能说明中含有输入条件的组合，最好在一开始就用**因果图法**，然后再按以上(1)、(2)、(3)步聚进行
  (5) 对照程序逻辑，检查设计方案。可根据程序可靠性方面的要求采用不同的**逻辑覆盖标准**。

90

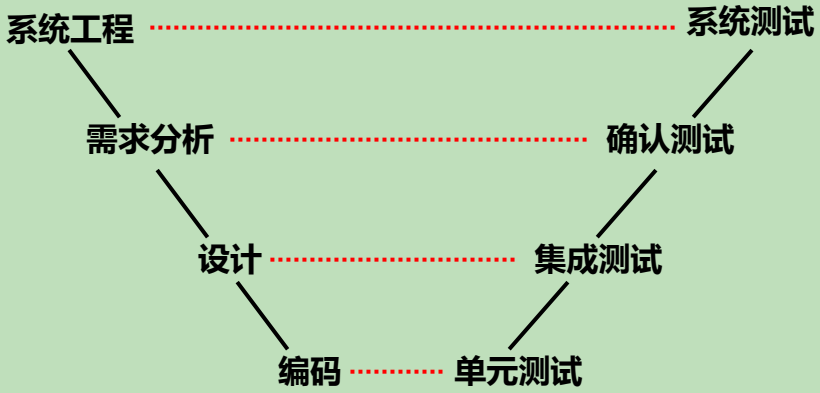
测试步骤

大型软件系统的测试主要有四个步骤：

- 单元测试
- 集成测试（组装测试）
- 确认测试
- 系统测试

91

- **V模型：描述软件开发各阶段与测试策略之间的对应关系。**



92

单元测试 (Unit Testing)

- 又称**模块测试**，**着重对软件设计的最小单元（软件构件或模块）进行验证**
- 根据设计描述，对重要的控制路径进行测试，以发现构件或模块内部的错误
- 通常采用白盒测试，多个构件或模块可以并行测试

93

单元测试的内容（1）

- **模块接口**：确保模块的输入/输出参数信息是正确的。
  - 如参数的个数、次序、类型等。
- **局部数据结构**：确保临时存储的数据在算法执行的整个过程中都能维持其完整性。
  - 如：不合适的类型说明、不同数据类型的比较或赋值、超越数据结构的边界等。
- **边界测试**：确保程序单元在极限或严格的情况下仍能正确地执行。

94

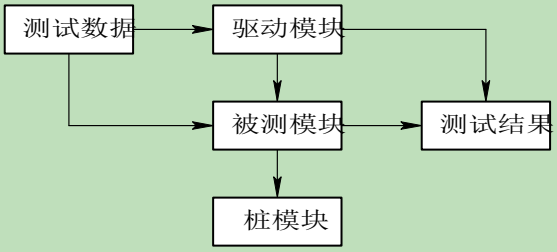
单元测试的内容（2）

- **重要执行路径**：确保模块中的所有语句都至少执行一次。
  - 常见的错误有：不正确的操作优先级、不同类型数据间的操作、不正确的初始化等。
- **错误处理**：对所有的错误处理路径进行测试。
  - 潜在的错误有：报错信息不足，无法确定错误的性质及位置、报错信息与真正的错误不一致、错误条件在错误处理之前就已引起系统异常等。

95

单元测试步骤

- 按下图配置测试环境
- 编写测试数据
- 进行多个单元的并行测试



•**驱动模块**：相当于所测模块的主程序，主要用来接收测试数据，启动被测模块，打印测试结果。

•**桩模块**：接收被测试模块的调用和输出数据，是被测模块的调用模块。

96

集成测试

- 也称**组装测试**、**联合测试**
- 按设计要求把通过单元测试的各个模块组装在一起进行测试，以发现**与接口有关的各种错误**
- 分类
  - 非渐增式测试：把所有模块结合起来测试
  - 渐增式测试：每次增加一个模块进行测试

97

集成测试方式

集成的方式有两种：

- **非渐增式集成：**将所有经过单元测试的模块组合在一起，然后对整个程序进行测试。
  - **发现错误时，很难为错误定位。**
  - **改正错误时，易引入新错误，新旧错误混在一起，更难定位。**
- **渐增式集成：**根据程序结构图，按某种次序挑选一个（或一组）尚未测试过的模块，把它集成到已测试好的模块中一起进行测试，每次增加一个（或一组）模块，直至所有模块全部集成到程序中。
  - **发现的错误，往往与新加入的模块有关**

98

渐增式测试

模块结合到软件的两种方式

- **自顶向下结合：**从主控制模块(“主程序”)开始，沿着软件的控制层次向下移动，从而逐渐把各个模块结合起来。
- **自底向上结合**：从软件结构最低层的模块开始组装和测试，当测试到较高层模块时，所需的下层模块均已具备，因而不再需要桩模块。

99

自顶向下集成

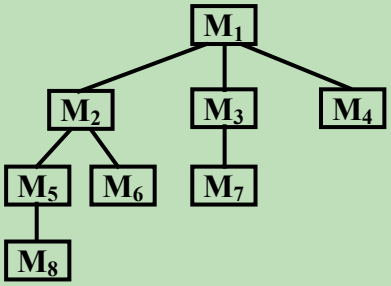
把主控模块所属的那些模块都装配到结构中去，有两种方法可供选择

- **深度优先测试次序：**

  M_1、M_2、M_5、M_8、M_6、M_3、M_7、M_4

- **广度优先测试次序：**

  M_1、M_2、M_3、M_4、M_5、M_6、M_7、M_8



100

自顶向下集成步骤

（1）主控模块（主程序）被直接用作驱动程序，所有直接从属于主控模块的模块用桩模块替换，对主控模块进行测试；

（2）根据集成的实现方式（深度优先或广度优先），下层的桩模块一次一个地替换成真正的模块，从属于该模块的模块用桩模块替换，然后对其进行测试；

（3）用**回归测试**来保证没有引入新的错误；

（4）重复第（2）和第（3）步，直至所有模块都被集成。

101

自顶向下集成

优点：

- 不需要驱动模块；
- 能尽早对程序的主要控制和决策机制进行检验，能较早发现整体性错误；
- 深度优先的自顶向下集成能较早对某些完整的程序功能进行验证。

缺点：

- 测试时低层模块用桩模块替代，不能反映真实情况；
- 重要数据不能及时回送到上层模块。

102

自底向上集成步骤

（1）将低层模块组合成能实现软件特定功能的**簇**；

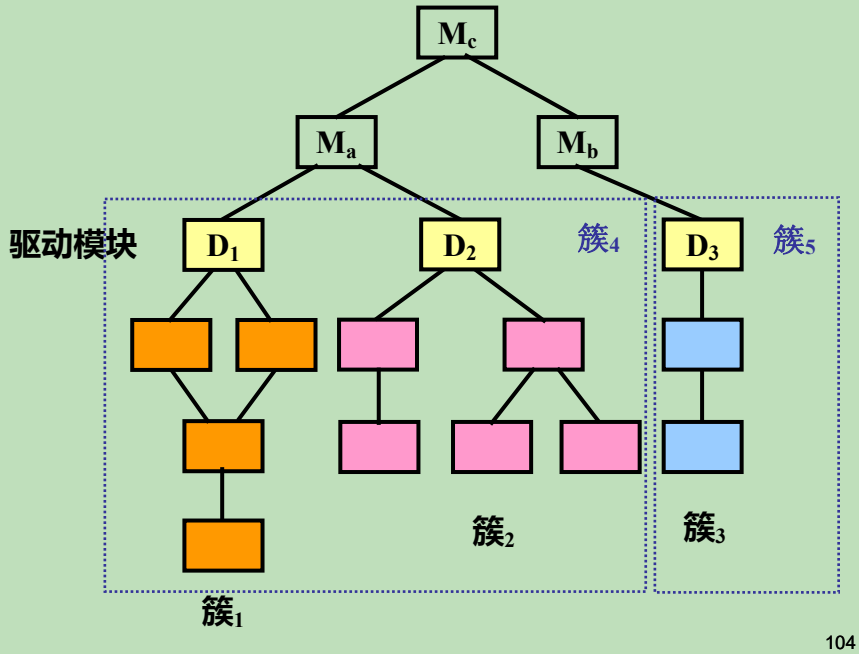
（2）为每个簇编写驱动程序，并对簇进行测试；

（3）移走驱动程序，用簇的直接上层模块替换驱动程序，然后沿着程序结构的层次向上组合新簇；

（4）对新簇测试后，都要进行回归测试，以保证没有引入新错误；

（5）重复第（2）步至第（4）步，直至所有的模块都被集成。

103



104

自底向上集成

优点：

- 不需要桩模块，容易组织测试；
- 整个程序结构分成若干个簇，同一层次的簇可并行测试，可提高效率。

缺点：

- 整体性的错误发现得较晚。

105

确认测试

- 又称**有效性测试**、**合格测试**或**验收测试**。
- 以软件**需求规约**为依据，主要发现软件与需求不一致的错误。
  - 检查软件是否实现了规约规定的全部功能要求
  - 文档资料是否完整、正确、合理
  - 其他的需求，如可移植性、可维护性、兼容性、错误恢复能力等是否满足

106

确认测试

确认测试的结果可分为两类：

- 满足需求规约要求的功能或性能特性，用户可以接受。
- 发现与需求规约有偏差，此时需列出问题清单。

107

配置复审

- 配置复审是确认测试的另一个重要环节。
- 目的：
  确保程序和文档配置齐全、分类有序，两者要一致，并包括了软件维护所必须的细节。

108

α测试和β测试

- **α测试**：由一个用户在开发环境下进行测试，也可以是开发机构内部的人员在模拟实际操作环境下进行的测试。
  - **经α测试后的软件称为β版软件**
- **β测试**：由软件的多个用户在一个或多个用户的实际使用环境下进行的测试。
  - 开发者一般不在现场

109

系统测试

- **系统测试**：将通过确认测试的软件，作为整个基于计算机系统的一个元素，与计算机硬件、外设、某些支持软件、数据和人员等其他系统元素结合在一起，在实际运行(使用)环境下，对计算机系统进行一系列的**组装测试**和**确认测试**。
- 常用的系统测试包括：
  - **恢复测试**
  - **安全测试**
  - **强度测试**
  - **性能测试**

110

系统测试

- **恢复测试：**检查系统的容错能力。
  - 在指定时间间隔内修正错误并重启系统。
- **安全性测试**：检验系统中已存在的系统安全性措施、保密性措施是否发挥作用，有无漏洞。
- **强度测试**：在一种需要**非正常**数量、频率或容量的方式下执行系统，检查系统对非正常情况的承受程度。
- **性能测试**：测试软件在被组装进系统的环境下运行时的性能。

111

补充内容

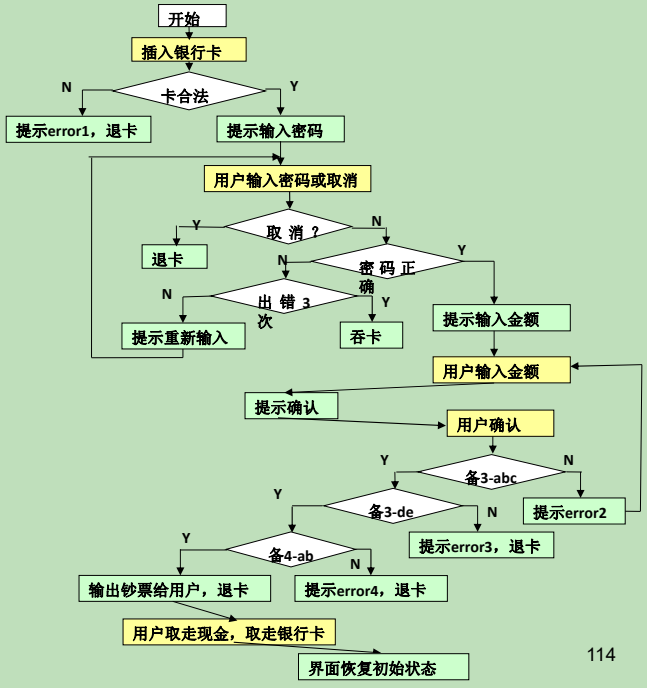
112

ATM取款机测试分析

- 基本事件流
  - 插入银行卡
  - 输入该银行卡的密码
  - 输入取钱金额
  - 系统同步银行主机
  - 用户提款，银行卡自动退出

113

ATM机场景



114

等价类划分

输入条件	有效等价类	无效等价类
银行卡	银行卡	非银行卡
密码	字符串为0～9之间的阿拉伯数字组合，密码长度为6位	长度不是6位的0~9之间的组合
金额	以50为单位，50~1500 RMB，单笔取款额最高为1500RMB；每24小时之内，取款的最高限额是4500RMB	非50的倍数，或大于1500，24小时内取款超过4500
确认	TRUE	
取现金	TRUE、FALSE	
取银行卡	TRUE、FALSE	

115

边界值分析

输入	内点	上点	离点
密码	000001 、999998	000000 、999999	00000 、1000000
金额	100、1350	50、1500	0、1550

116

测试用例（第一组）

测试用例编号	ATM_ST_FETCH_001
测试项目	银行ATM机取款
测试标题	输入合法密码和金额，按金额确认，并取走现金和银行卡，内点小
重要级别	高
预置条件	系统存在该用户
输入	金额100，密码000001
操作步骤	1、插入银行卡；2、输入密码000001；3、输入金额100；4、点击确定；5、取走现金；6、取走银行卡。
预期输出	1、提示输入密码；2、提示输入金额；3、提示确认；4、输出钞票；5、退出银行卡；6、界面恢复初始状态

117

测试用例（第二组）

测试用例编号	ATM_ST_ FETCH _002
测试项目	银行ATM机取款
测试标题	输入合法密码和金额，按金额确认，不取走现金和银行卡，内点大
重要级别	中
预置条件	系统存在该用户
输入	金额1350，密码999998
操作步骤	1、插入银行卡；2、输入密码999998；3、输入金额1350；4、点击确定；5、不取走现金；6、不取走银行卡。
预期输出	1、提示输入密码；2、提示输入金额；3、提示确认；4、输出钞票；5、退出银行卡；6、界面恢复初始状态

118

测试用例（第三组）

测试用例编号	ATM_ST_ FETCH _003
测试项目	银行ATM机取款
测试标题	输入合法密码和金额，按金额确认，并取走现金和银行卡，上点小。
重要级别	中
预置条件	系统存在该用户
输入	金额50，密码000000
操作步骤	1、插入银行卡；2、输入密码000000；3、输入金额50；4、点击确定；5、取走现金；6、取走银行卡。
预期输出	1、提示输入密码；2、提示输入金额；3、提示确认；4、输出钞票；5、退出银行卡；6、界面恢复初始状态。

119

测试用例（第四组）

测试用例编号	ATM_ST_ FETCH _004
测试项目	银行ATM机取款
测试标题	输入合法密码和金额，按金额确认，并取走现金和银行卡，上点大。
重要级别	中
预置条件	系统存在该用户
输入	金额1500，密码999999
操作步骤	1、插入银行卡；2、输入密码999999；3、输入金额1500；4、点击确定；5、取走现金；6、取走银行卡。
预期输出	1、提示输入密码；2、提示输入金额；3、提示确认；4、输出钞票；5、退出银行卡；6、界面恢复初始状态。

120

测试用例（第五组）

测试用例编号	ATM_ST_ FETCH _005
测试项目	银行ATM机取款
测试标题	插入非银行卡
重要级别	中
预置条件	
输入	
操作步骤	插入IC卡
预期输出	提示用户“您使用的银行卡无效！”，3秒钟后，自动退出该银行卡。

121

测试用例（第六组）

测试用例编号	ATM_ST_ FETCH _006
测试项目	银行ATM机取款
测试标题	输入非法密码，离点小
重要级别	中
预置条件	系统存在该用户
输入	密码00000
操作步骤	1、插入银行卡；2、输入密码00000。
预期输出	1、提示输入密码； 2、提示用户“您输入的密码无效，请重新输入”；

122

测试用例（第七组）

测试用例编号	ATM_ST_ FETCH _007
测试项目	银行ATM机取款
测试标题	输入非法密码，离点大
重要级别	中
预置条件	系统存在该用户
输入	密码1000000
操作步骤	1、插入银行卡；2、输入密码1000000。
预期输出	1、提示输入密码； 2、提示用户“您输入的密码无效，请重新输入”；

123

测试用例（第八组）

测试用例编号	ATM_ST_ FETCH _008
测试项目	银行ATM机取款
测试标题	输入非法金额，离点小
重要级别	中
预置条件	系统存在该用户
输入	密码123456，金额为0
操作步骤	1、插入银行卡；2、输入密码123456；3、输入金额0。
预期输出	1、提示输入密码；2、提示输入金额；3、提示用户“您输入的提款金额错误，请输入以50为单位的金额”；

124

测试用例（第九组）

测试用例编号	ATM_ST_ FETCH _009
测试项目	银行ATM机取款
测试标题	输入非法金额，离点大。
重要级别	中
预置条件	系统存在该用户
输入	密码123456，金额为1550
操作步骤	1、插入银行卡；2、输入密码123456；3、输入金额1550。
预期输出	1、提示输入密码；2、提示输入金额；3、提示用户 “您输入的金额错误，单笔提款上限金额是1500RMB，请重新输入”；

125

测试用例（第十组）

测试用例编号	ATM_ST_ FETCH _010
测试项目	银行ATM机取款
测试标题	提取金额达到上限
重要级别	中
预置条件	系统存在该用户
输入	密码123456，金额为1500，50
操作步骤	1、插入银行卡；2、输入密码123456；3、输入金额1500；4、且在23小时内，提款4500；5、在23小时59分，提款50。
预期输出	1、提示输入密码；2、提示输入金额；3、提示用户 “24小时内只能提取4500RMB，请重新输入提款金额”

126

测试用例（第十一组）

测试用例编号	ATM_ST_ FETCH _011
测试项目	银行ATM机取款
测试标题	插入卡后取消操作
重要级别	底
预置条件	无
输入	无
操作步骤	1、插入银行卡；2、点击取消。
预期输出	1、提示输入密码；2、退出银行卡。

127

测试用例（第十二组）

测试用例编号	ATM_ST_ FETCH _012
测试项目	银行ATM机取款
测试标题	输入非法密码
重要级别	中
预置条件	系统存在该用户
输入	密码111111
操作步骤	1、插入银行卡；2、输入密码111111；3、重复操作“步骤2次（一共出错3次）。
预期输出	1、提示输入密码；2、提示用户“您输入的密码无效，请重新输入”；3、重复“步骤“步骤2次（总共提示3次出错）；4、系统吞卡。

128

测试用例（第十三组）

测试用例编号	ATM_ST_ FETCH _013
测试项目	银行ATM机取款
测试标题	输入提款金额大于账户内金额
重要级别	中
预置条件	系统存在该用户，账户存款为1000
输入	密码123456，金额为1500
操作步骤	1、插入银行卡；2、输入密码123456；3、输入金额1500；4、点击确定；
预期输出	1、提示用户“抱歉，您的存款余额不足！”；2、3秒钟后，自动退出银行卡。

129

谢谢！

130