软件测试

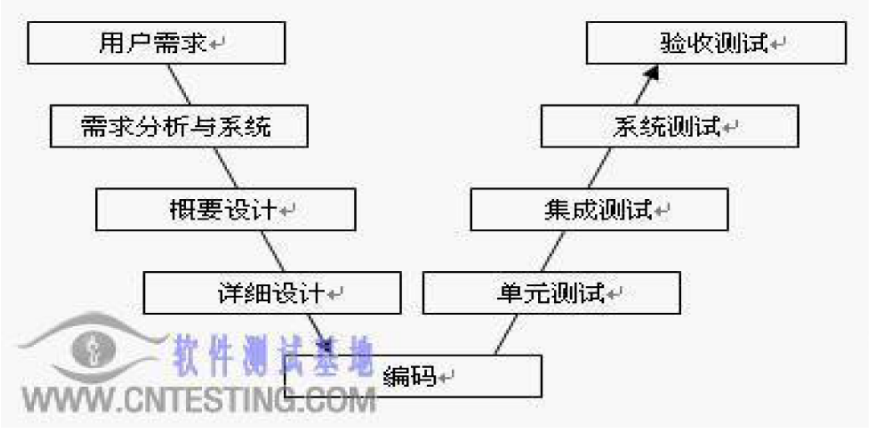
软件测试：ISO9000 定义：测试是一种基于机器的，对代码执行测试，确认测试的活动。

软件测试是为了发现错误而执行程序的过程。

软件缺陷：结果出错，功能失效，与用户需求不一样。

软件测试的模型：V 模型，W 模型，H 模型.

V 模型是建立在编码的基础上



软件测试的分类

- 按照开发阶段划分（**A** 卷）
  - *单元测试*：模块测试，检查每个程序单元嫩否正确实现详细设计说明中的模块功能等。
  - *集成测试*：组装测试，将所有的程序模块进行有序、递增的测试，检验程序单元或部件的接口关系
  - *系统测试*：检查完整的程序系统能否和系统（包括硬件、外设和网络、系统软件、支持平台等）正确配置、连接，并满足用户需求。
  - *确认测试*：证实软件是否满足特定于其用途的需求，

是否满足软件需求说明书的规定。

- *验收测试*：按照项目任务或合同，供需双方签订的验收依据文档进行的对整个系统的测试与评审，决定是否接受或拒收系统。

- 按照测试技术划分（**B** 卷）
  - 白盒测试：通过对程序内部结构的分析、检测来寻找问题。检查是否所有的结构及逻辑都是正确的，检查软件内部动作是否按照设计说明的规定正常进行。**--**结构测试
  - 黑盒测试：通过软件的外部表现来发现错误，是在程序界面处进行测试，只是检查是否按照需求规格说明书的规定正常实现。
  - 灰盒测试：介于白盒测试与黑盒测试之间的测试，关注输出对输入的正确性；同时，也关注内部表现，不像白盒那样详细，只是通过一些表征性现象、事件、标志来判断内部的运行状态。

<u>软件测试的原则（简答题）</u>

- **1** 完全测试的不可能性
- **2** 软件测试是有风险的活动
- **3** 测试无法显示潜伏的软件缺陷和故障
- **4.** 充分注意测试中的群集现象
- **5.**杀虫剂现象

- **6.**并非所有的软件缺陷都要修复
- 7.难以描述的软件缺陷
- 80-20 原则（第一个含义：**80%** 的软件缺陷常常生存在软件 **20%** 的空间里。）

测试用例的定义：测试用例是为了特定的用户而设计的一组测试输入，执行条件和预期结果。

黑盒测试用例设计的几种方法

- （一）等价类划分法
- （二）边界值分析法
- （三）决策表法
- （四）因果图法
- （五）场景法

等价类的划分原则

按照区间划分——在输入条件规定了取值范围或值的个数的情况下，可以确定一个有效等价类和两个无效等价类。

按照数值划分——在规定了一组输入数据（假设包括 **n** 个输入值），并且程序要对每一个输入值分别进行处理的情况下，可确定 **n** 个有效等价类（每个值确定一个有效等价类）和一个无效等价类（所有不允许的输入值的集合）。

按照数值集合划分——在输入条件规定了输入值的集合或规定了“必须如何”的条件下，可以确定一个有效等价类和一个无效等价类（该集合有效值之外）。

按照限制条件或规则划分——在规定了输入数据必须遵守的规则或限制条件的情况下，可确定一个有效等价类（符合规则）和若干个无效等价类（从不同角度违反规则）。

怎样用边界值分析法设计测试用例？

（**1**）首先确定边界情况。通常输入或输出等价类的边界就是应该着重测试的边界情况。

（**2**）选取正好等于、刚刚大于或刚刚小于边界的值作为测试数据，而不是选取等价类中的典型值或任意值。

标准测试：4N+1

强壮测试：6N+1

例题：有二元函数 ***f(x,y)****，其中* ***x∈[1,12]，y∈[1,31]****。则采用边界值分析法设计的测试用例是：*

{ <1,15>, <2,15>, ><11,15>, <12,15>, <6,15>, <6,1>, <6,2>, <6,30>, <6,31>, }

2.NextDate *函数包含三个变量* ***month***、***day*** *和* ***year***，*函数的输出为输入日期后一天的日期。要求输入变量* ***month***、***day*** *和* ***year*** *均为整数值，并且满足下列条件：*

条件 ***1 1≤ month ≤12***

条件 ***2 1≤ day ≤31***

条件 ***3 1912≤ year ≤2050***

month 变量的有效等价类：

M1: {month=4,6,9,11} **M2: {month=1,3,5,7,8,10}**

M3: {month=12} **M4: {month=2}**

- **day** 变量的有效等价类：

D1: {1≤day≤27} **D2: {day=28}** **D3: {day=29}**

D4: {day=30} **D5: {day=31}**

- **year** 变量的有效等价类：

Y1: {year 是闰年**}** **Y2: {year** 不是闰年**}**

- 程序中可能采取的操作有以下六种：

a1: 不可能 **a2: day**＋**1** **a3: day=1**

a4: month+1 **a5: month=1** **a6: year+1**

3.

决策表的概念：决策表是分析和表达多逻辑条件下执行不同操作情况的工具。

决策表的优点

- 能够将复杂的问题按照各种可能的情况全部列举出来，简明并避免遗漏。因此，利用决策表能够设计出完整的测试用例集合。
- 最为严格，最具逻辑性的测试方法。

决策表的组成

- 决策表通常由以下 **4** 部分组成：
  - 条件桩—列出问题的所有条件

- 条件项—针对条件桩给出的条件列出所有可能的取值
- 动作桩—列出问题规定的可能采取的操作
- 动作项—指出在条件项的各组取值情况下应采取的动作

决策表的生成----构造决策表的 5 个步骤：

1）列出所有的条件桩和动作桩。

（**2**）确定规则的个数。

（**3**）填入条件项。

（**4**）填入动作项，得到初始决策表。

（**5**）简化决策表，合并相似规则。

- 若表中有两条以上规则具有相同的动作，并且在条件项之间存在极为相似的关系，便可以合并。
- 合并后的条件项用符号“**-**”表示，说明执行的动作与该条件的取值无关，称为无关条件。

因果图法设计测试用例思想

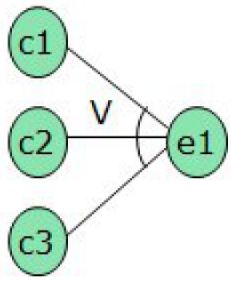
- 首先从程序规格说明书的描述中,找出因**(**输入条件**)**和果**(**输出结果或者程序状态的改变**)**,
- 然后通过因果图转换为判定表,最后为判定表中的每一列设计一个测试用例.

四种关系

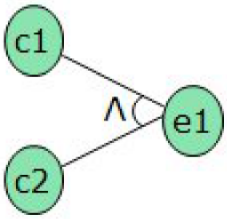
- 恒等：若 **c1** 是 **1**，则 **e1** 也为 **1**，否则 **e1** 为 **0**；

- 非：若 **c1** 是 **1**，则 **e1** 为 **0**，否则 **e1** 为 **1**；用符号“**~**”表示。

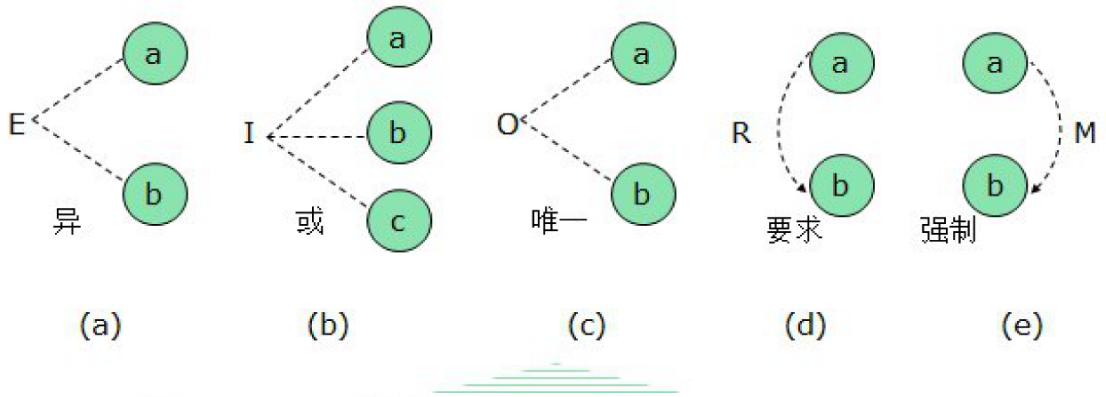
或：若 **c1** 或 **c2** 或 **c3** 是 **1**，则 **e1** 是 **1**，否则 **e1** 为 **0**，“或”可有任意个输入；用符号“**V**”表示。



- 与：若 **c1** 和 **c2** 都是 **1**，则 **e1** 为 **1**，否则 **e1** 为 **0**，“与”也可有任意个输入。用符号“∧”表示。



在实际问题当中输入状态相互之间还可能存在某些依赖关系，称为“约束”



» **E** 约束（异）：**a** 和 **b** 中最多有一个可能为 **1**，即 **a** 和 **b** 不能同时为 **1**；

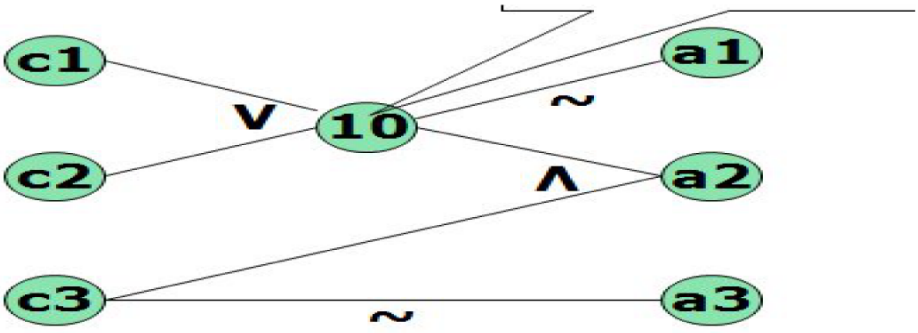
» **I** 约束（或）：**a**、**b**、**c** 中至少有一个必须是 **1**，即 **a**、**b**、**c** 不能同时为 **0**；

» **O** 约束（唯一）：**a** 和 **b** 必须有一个且仅有一个为 **1**；

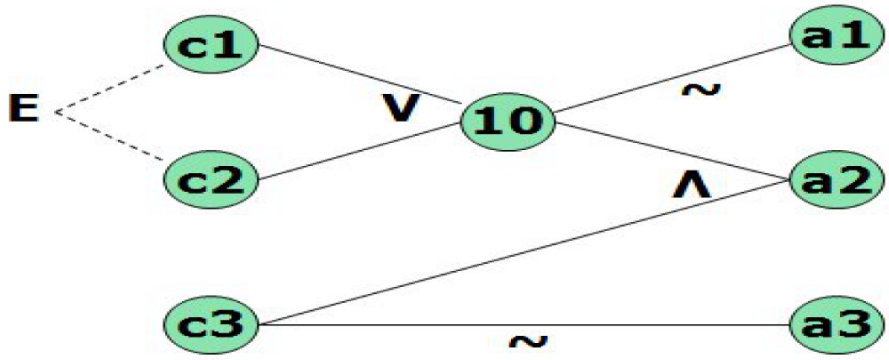
» **R** 约束（要求）：**a** 是 **1** 时，**b** 必须是 **1**；

» **M** 约束（强制）：若结果 **a** 是 **1**，则结果 **b** 强制为 **0**。

例题：程序的规格说明要求：输入的第一个字符必须是“**#**”或“*****”，第二个字符必须是一个数字，在此情况下进行文件的修改；如果第一个字符不是“**#**”或“*****”，则给出信息 **N**；如果第二个字符不是数字，则给出信息 **M**。



因果图表示



具有E约束的因果图表示

	1	2	3	4	5	6	7	8
C1	1	1	1	1	0	0	0	0
C2	1	1	0	0	1	1	0	0
C3	1	0	1	0	1	0	1	0
11			1	1	1	1	0	0
a1							√	√
a2			√		√			
a3				√		√		√
不可能	√	√						
测试用例			#3	#B	*7	*M	C2	CM

白盒测试

程序的结构形式是白盒测试的重要依据。

白盒测试的方法

1.逻辑覆盖法

2.基本路径法

主要是测试覆盖率，以程序内在逻辑结构为基础的测试。包括以下 **6** 种类型：

- 语句覆盖
- 判定覆盖
- 条件覆盖

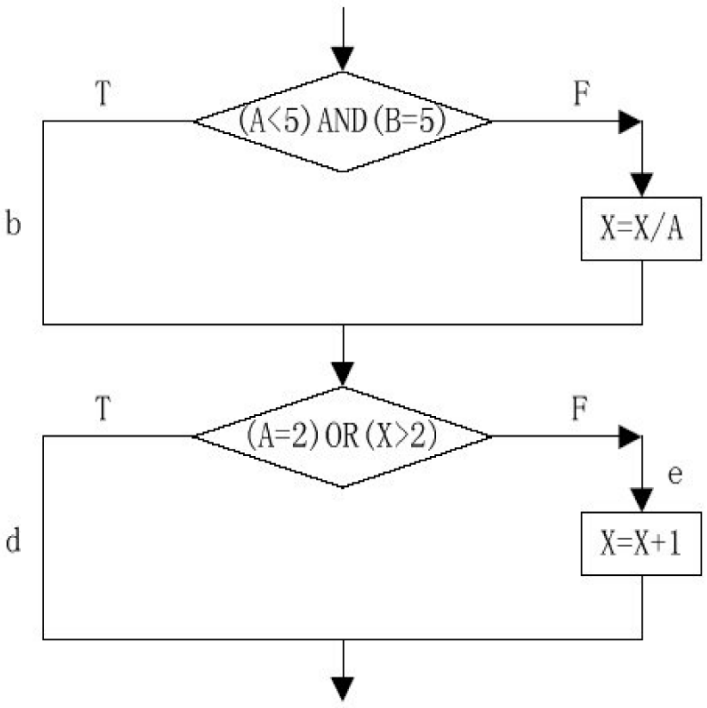
- 判定-条件覆盖
- 条件组合覆盖
- 修正-判定条件覆盖

基本路径测试

为了清晰描述这种白盒测试方法，需要首先对有关白盒测试的几个基本概念进行说明：

- 流图
- 环形复杂度
- 图矩阵

考



根据左图给出的程序流程图，完成以下要求：

（1）画出相应的控制流图。

（2）计算环形复杂度。

（3）给出相应的图矩阵。

（4）找出程序的独立路径集合。

循环测试

⦿ 循环分为 **4** 种不同类型：

- 简单循环
- 嵌套循环
- 连锁循环（串接循环）
- 非结构循环（不规则循环）

⦿ **(1)** 简单循环测试

①零次循环：从循环入口到出口

②一次循环：检查循环初始值

③二次循环：两次通过循环

④***m*** 次循环：检查多次循环

⑤最大次数循环 **n**、比最大次数多一次 **n+1**、少一次的循环 **n-1**。

⦿ 程序插桩技术

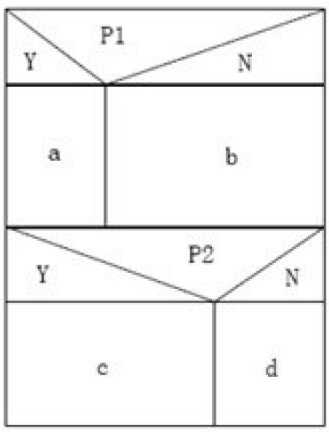
方法简介

如果我们想要了解一个程序在某次运行中所有可执行语句被覆盖的情况，或是每个语句实际执行次数,最好的办法就是利用程序插桩技术.

最少测试用例数计算

例如，下图表达了两个顺序执行的分支结构。当两个分支谓词

P1 和 P2 取不同值时，将分别执行 a 或 b 及 c 或 d 操作



显然，要测试这个小程序，需要至少提供 4 个测试用例才能作到逻辑覆盖，使得 ac、ad、bc 及 bd 操作均得到检验。其实，这里的 4 是图中的第 1 个分支谓词引出的两个操作，及第 2 个分支谓词引出的两个操作组合起来而得到的，即 2×2=4。并且，这里的 2 是由于两个并列的操作，即 1+1=2 而得到的。

单元测试的定义

- ◉ 单元测试又称模块测试，是针对软件设计的最小可测试单元，进行正确性检验的测试工作。，其依据是详细设计描述，对模块内所有重要的控制路径设计测试用例，以便发现模块内部的错误。
- ◉ 单元测试多采用白盒测试技术，系统内多个模块可以并行地进行单元测试。

单元测试的对象

单元具有一些基本属性，如：明确的功能、规格定义，与其他部分明确的接口定义等，可清晰的与同一程序的其他单元划分开来。

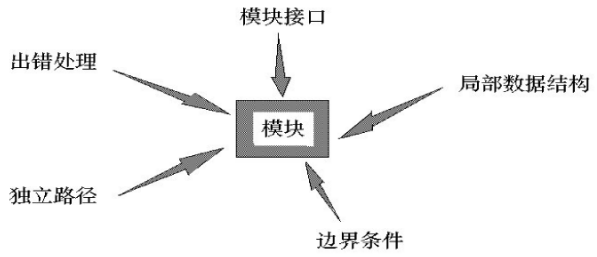
- ◉ 结构化程序：一个函数或子过程（eg：C 语言中）
- ◉ 面向对象的程序：一个类或类的方法（eg：Java、C++语言中）

一个窗口或一个菜单（eg：图形化软件中）

什么时候进行单元测试

- ◉ 通常在编码阶段进行
- ◉ ExtremeProgramming（极限编程，简称 XP）是由 KentBeck 在 1996 年提出。
- ◉ XP 提倡在开始写程序之前先写单元测试。开发人员应该经常把开发好的模块整合到一起，每次整合后都要运行单元测试；做任何的代码复核和修改，都要运行单元测试；发现了 BUG，就要增加相应的测试。

单元测试内容



单元测试一般步骤

1)编译运行程序

- ◉ 首先编译程序，没有语法上的错误，编译通过；
- ◉ 然后运行程序，输入 1 2 3 4 0，按回车；
- ◉ 输出 1 2 3 4 1，符合预期结果。

静态测试

检查程序中不符合编码规范的地方

C/C++编码规范

动态测试

- 深入检查代码的正确性，非法数据的容错性和边界值等问题。
  - 边界值
  - 错误处理
  - 非法数据的容错性