高级软件工程

软件测试

沈立炜

shenliwei@fudan.edu.cn

软件测试概述

软件测试的概念

- **正向的维度**
  - 软件测试就是以评价一个程序或系统的质量或能力为目的的一项活动 [Bill Hetzel]
  - 软件测试的目的是验证软件系统的正确性
- **反向的维度**
  - 软件测试是以发现错误为目的而执行一个程序或系统的过程 [Glenford J. Myers]
  - 测试的目地是为了证明程序是有错误的；测试人员需要以“破坏性”的手段来找到系统中不符合要求的地方

正向维度 VS. 反向维度

- 正向维度的软件测试从质量保证的角度来指导测试工作，这对于安全攸关的软件系统而言是必要的
- 反向维度的软件测试考虑测试工作的目标与效率，这种测试用于一般的软件系统：软件质量只需维持在一个用户可以接受的水平即可，而不用过分追求与安全攸关系统相同的质量水准

IEEE 729-1983 定义

使用人工或自动的手段来运行或测量软件系统的过程，目的是检验软件系统是否满足规定的要求，并找出与预期结果之间的差异。

软件测试

狭义 VS. 广义的软件测试

- **狭义的软件测试**
  - 测试的测试对象局限在软件代码之上
  - 依赖于测试人员运行待测软件来验证系统的功能正确性或发现程序缺陷
  - IEEE软件工程知识体系中，软件测试就被定义为一个动态的过程，它基于一组有限的测试用例执行待测程序，从而验证一个程序是否提供了预期的行为
- **广义的软件测试**
  - 需要向前延伸至软件文档（需求文档、设计文档等）、模型等一系列软件开发过程的中间制品
  - 测试已不再是程序编码之后的一个环节，而应是贯穿于整个软件生存周期之中的一系列活动

验证 VS. 确认

- **验证（verification）**
  - 目的在于确保已实现的软件符合其产品规格说明书所定义的系统功能和特性
  - 在软件生存周期的各个阶段，可以用下一个阶段的制品来检查产品或中间产品是否满足上一个阶段的规格定义，即确保开发者正在正确地开发产品（build the product right）
- **确认（validation）**
  - 目的是证明软件或构件在用户环境下能够实现用户的真实需求，即确保已开发出的软件能够符合用户的真正意图
  - 与验证相比，确认是确保开发者正在开发正确的产品（build the right product）

典型的软件测试级别

- 单元测试（Unit Testing）
- 集成测试（Integration Testing）
- 系统测试（System Testing）
- 验收测试（Acceptance Testing）
- 回归测试（Regression Testing）

软件测试的原则

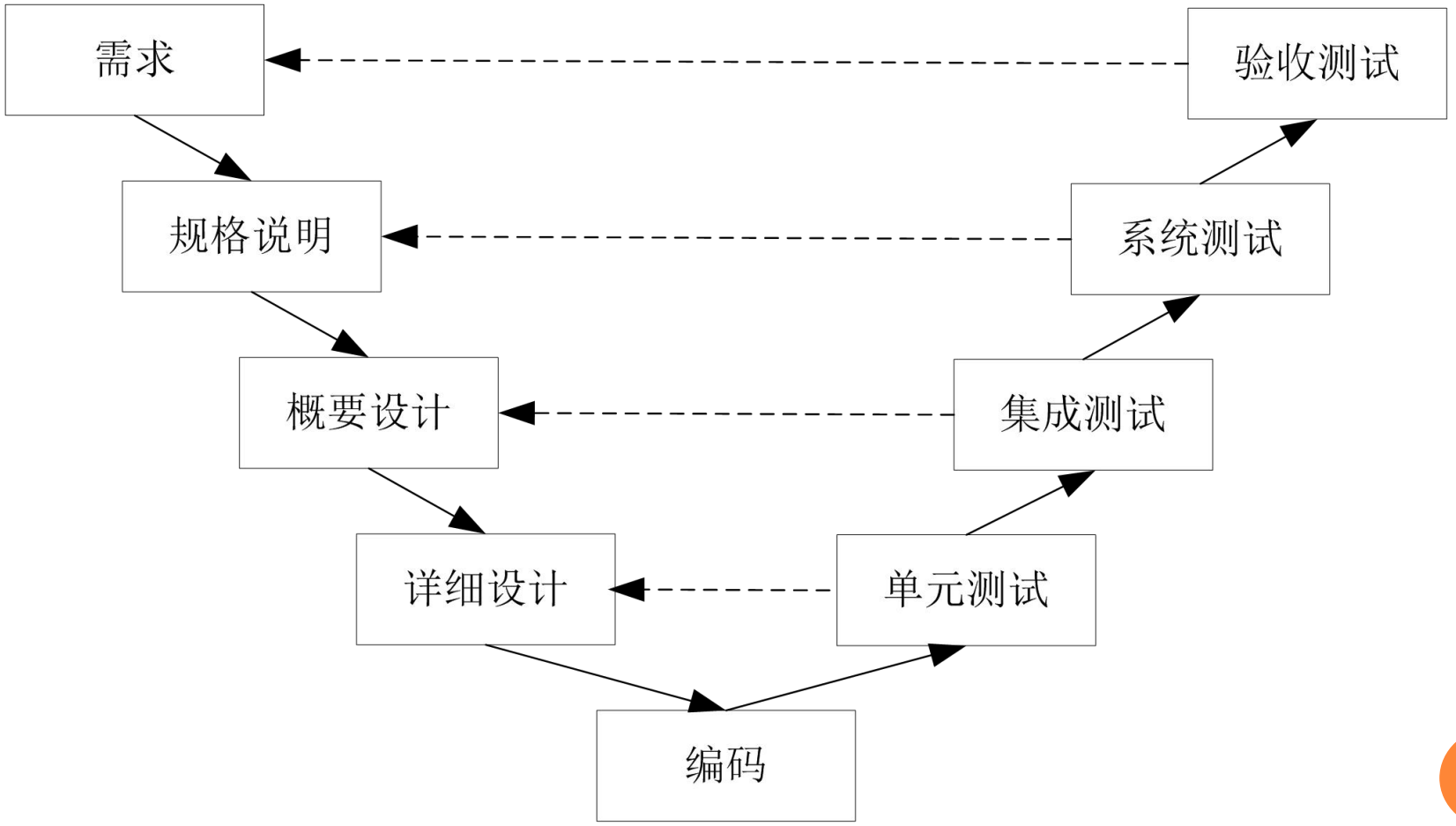
- “足够好”（good enough）
- Pareto原则
- 测试贯穿于软件生存周期
- 尽早测试
- 避免同化效应

软件测试过程

软件测试过程模型

- V模型
- W模型
- H模型

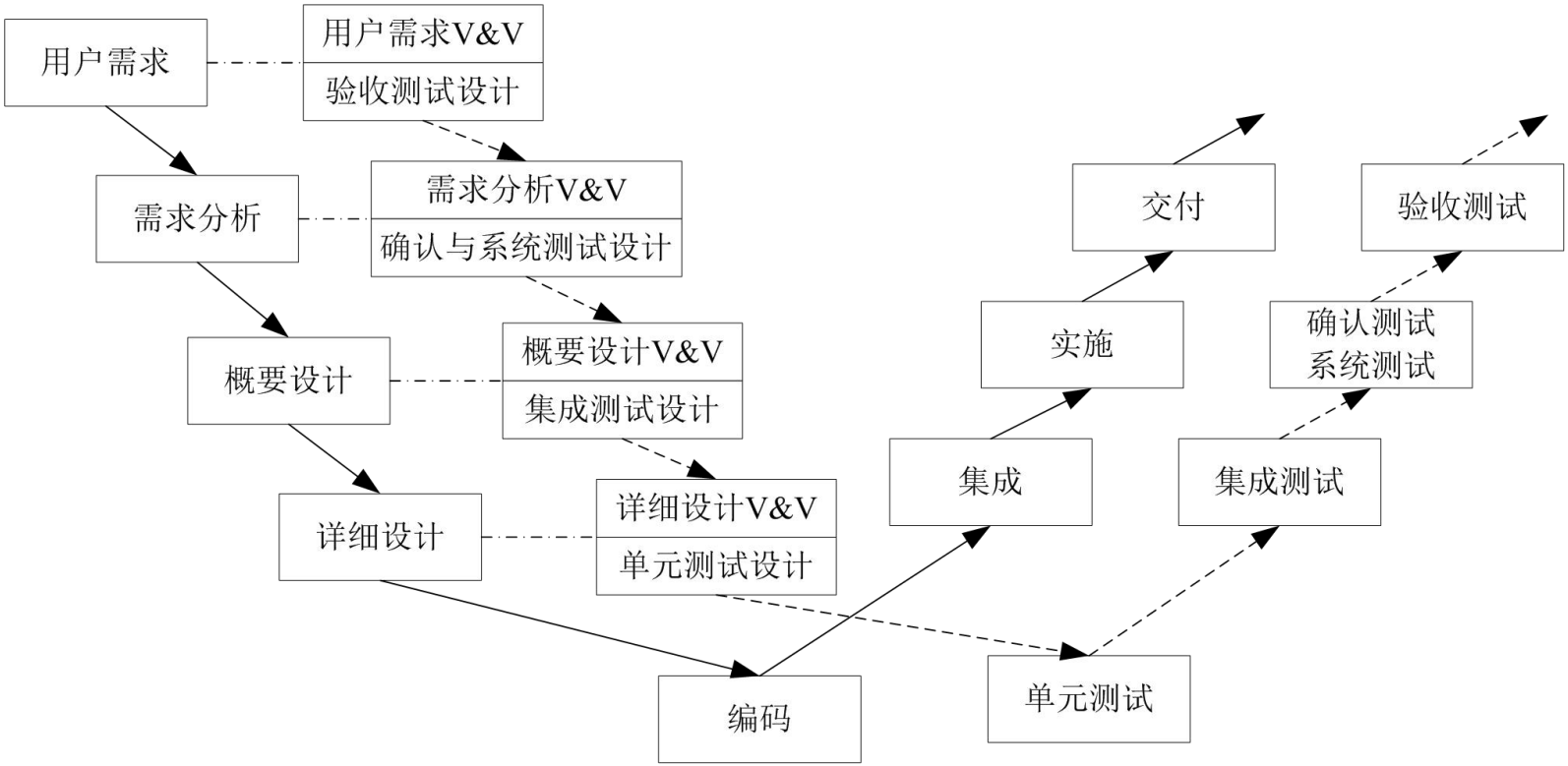
V模型



V模型的特点和局限性

- **特点**
  - 将测试活动划分为不同的阶段
  - 测试的执行都是在编码活动完成之后才进行的
  - 测试主要针对软件代码
  - 理想情况下，软件开发过程中的每一个阶段都为对应的测试预先设计了相应的测试用例
- **局限**
  - 将测试作为在需求分析、概要设计、详细设计及编码之后才被执行的活动，这会使得早期需求、设计阶段产生的缺陷只有在后期的测试活动中才能被发现，从而导致部分需求与设计的返工，增加软件开发的成本

W模型


用户需求
用户需求V&V
验收测试设计
需求分析
需求分析V&V
确认与系统测试设计
概要设计
概要设计V&V
集成测试设计
详细设计
详细设计V&V
单元测试设计
编码
单元测试
集成
集成测试
实施
确认测试
系统测试
交付
验收测试

特点

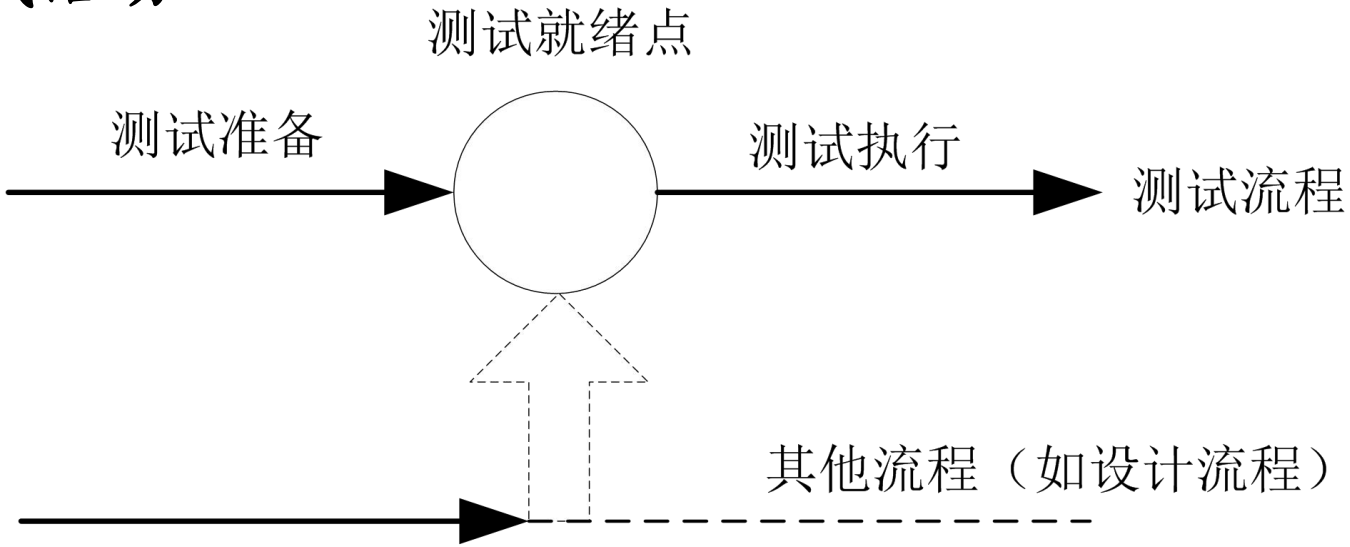
- 测试是伴随着整个软件的开发周期而进行的
- 测试的对象不仅仅是程序，也包括需求、设计等制品
- 测试人员在编码活动开始之前就可以对特定阶段生产的中间制品开展测试活动
- 有利于尽早发现软件开发过程中的缺陷，降低软件维护成本

局限

- 测试仍旧保持严格的顺序关系，只有当前一阶段完成后才能进入下一阶段
- 无法实现测试活动的迭代，即无法对之前的制品进行修改

H模型

- 将测试活动完全独立出来，形成一个完全独立的流程，并且清晰地划分了测试准备活动和测试执行活动
- 没有限定测试活动的先后顺序，因此整个软件的测试可以按照阶段顺序进行，也可以迭代地执行
- 只要测试条件成熟且测试准备活动完成时，就可执行测试活动



软件测试标准

- ISO/IEC 90003:2004 （Software engineering — Guidelines for the application of ISO 9001:2000 to computer software）
- “软件工程 GB/T 19003-2008应用于计算机软件的指南”
  - 在“设计和开发”章节，该标准包括“设计和开发评审”、“设计和开发验证”与“设计和开发确认”这三个与测试相关的活动，标准中给出了这些活动的内容以及执行这些活动所应遵循的原则。
  - 特别在“设计和开发确认”活动中，该标准进一步指明了测试的不同级别（单元测试、集成测试等），以及执行不同测试技术的准则。

ISO/IEC/IEEE 29119

- 定义与术语（ISO/IEC 29119-1: Concepts & Definitions）
- 测试过程（ISO/IEC 29119-2: Test Processes）
- 测试文档（ISO/IEC 29119-3: Test Documentation）
- 测试技术（ISO/IEC 29119-4: Test Techniques）
- 关键字驱动的测试（ISO/IEC 29119-5: Keyword Driven Testing）

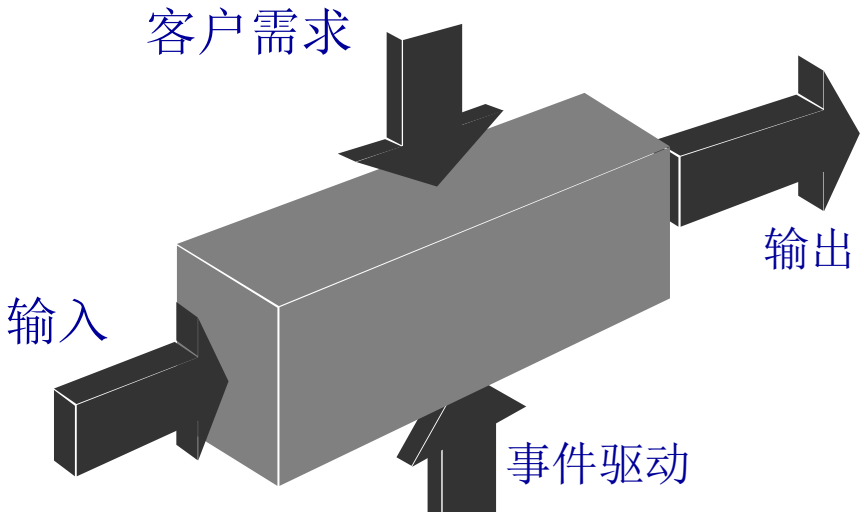
定义了三层的通用测试过程框架，该框架包含组织测试过程、测试管理过程（测试计划、测试监控、测试完成）以及动态测试过程（测试设计与实现、测试环境搭建与维护、测试执行、测试报告）。针对各个过程，该标准以状态图的方式详细描述了实现该过程的各项活动。

- **GB/T 15532-2008《计算机软件测试规范》**
  - 规定了计算机软件生存周期内各类软件产品的基本测试方法、过程和准则
  - 对软件测试过程中的测试对象和目的、测试的组合和管理、技术要求、测试内容、测试环境、测试方法、准入条件、准出条件、测试过程和输入文档等条目做出了要求

软件测试技术

黑盒测试

- 黑盒测试也称为功能测试或数据驱动的测试
- 测试人员无法查看同时也无需了解其中的代码与程序结构，他们只是依据程序规格说明，根据程序的输入、输出来验证程序功能的正确性



等价类划分

- 将程序的输入划分为一组等价类，对等价类中一个输入数据的测试结果能够等同于针对该等价类中其他输入数据的测试
- 步骤
  - 划分等价类
  - 编写测试用例

划分等价类

- 有效等价类是对程序而言合法的输入数据集合
- 无效等价类则是非法的输入数据集合

等价类划分原则-1

- 若输入条件规定了输入数据的取值范围或者个数，那么可以确定一个有效等价类和两个无效等价类
- 若输入条件规定了数据值的集合，或者是规定了“必须如何”的条件，那么可以确定一个有效等价类和一个无效等价类
- 若输入条件是一个布尔值，那么可以确定一个有效等价类和一个无效等价类

等价类划分原则-2

- 若程序规定了输入数据的一组值（假定有n个值），并且程序对每一个输入值进行不同处理，那么可以确定n个有效等价类和一个无效等价类
- 若程序规定了某个输入数据必须遵守的规则，那么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）
- 在确定已知等价类中各元素在程序处理方式不同的情况下，则应该再将该等价类进行划分，成为更小的等价类

确定测试用例

- 建立等价类表

输入条件	有效等价类	无效等价类
……	……	……

- 一个测试用例应尽可能多地覆盖尚未被覆盖的有效等价类，重复该步骤，直至所有的有效等价类都被覆盖为止；
- 一个测试用例应只覆盖一个尚未被覆盖的无效等价类，重复该步骤，直至所有的无效等价类都被覆盖为止

示例

- 一个对输入三角形进行分类的程序功能是：读入代表三角形边长的3个整数，即a、b、c，三条边的长度都在1至100之间（包括1和100），判定它们能否组成三角形，若不能，显示“输入错误”；若三边相等，显示“等边三角形”；若只有两边相等，显示“等腰三角形”；若三边各不相等，则显示“一般三角形”

三角形分类程序的输入等价类表

输入条件	有效等价类	无效等价类
3条边的赋值	(1) 0 < a < 101 (2) 0 < b < 101 (3) 0 < c < 101	(4) a <= 0　(5) a >= 101 (6) b <= 0　(7) b >= 101 (8) c <= 0　(9) c >= 101
构成一般三角形	(10) a < b + c (11) b < a + c (12) c < a + b	(13) a >= b + c (14) b >= a + c (15) c >= a + b
构成等边三角形	(16) a = b= c	
构成等腰三角形	(17) a = b, a != c (18) b = c, b != a (19) a = c, a!= b	

三角形分类程序的测试用例

编号	输入数据	预期输出	覆盖等价类
1	a=50, b=60, c=70	一般三角形	(1)(2)(3)(10)(11)(12)
2	a=50, b=50, c=50	等边三角形	(16)
3	a=50, b=50, c=10	等腰三角形	(17)
4	a=10, b=50, c=50	等腰三角形	(18)
5	a=50, b=10, c=50	等腰三角形	(19)
6	a=0, b=50, c=60	输入错误	(4)
7	a=102, b=50, c=60	输入错误	(5)
8	a=50, b=0, c=60	输入错误	(6)
9	a=50, b=102, c=60	输入错误	(7)
10	a=60, b=50, c=0	输入错误	(8)
11	a=60, b=50, c=102	输入错误	(9)
12	a=80, b=30, c=40	输入错误	(13)
13	a=30, b=80, c=40	输入错误	(14)
14	a=40, b=30, c=80	输入错误	(15)

边界值分析法

- 采用将某个变量输入范围边界上的值作为输入条件，从而验证系统功能是否正确的测试方法
- 所选择的测试数据一般位于输入的边界条件或临界值，以及这些边界条件、临界值附近的值

选择测试用例的技巧

- 若输入条件规定了值的范围，那么选择刚刚达到这个范围的边界值，以及刚刚超过这个范围边界的值
- 若输入条件规定了值的个数，那么选择最大个数、最小个数、比最大个数多1个、比最小个数少1个的数
- 若输入范围是一个有序的集合，例如有序表、顺序文件等，那么应选取集合的第一个和最后一个元素作为测试数据

三角形分类程序的边界值测试用例

编号	输入数据	预期输出
15	a=1, b=1, c=1	等边三角形
16	a=100, b=100, c=100	等边三角形
17	a=100, b=50, c=50	输入错误

判定表与因果图

- 将输入数据作为程序的参数，那么参数之间的影响，即参数的组合可能成为主要的错误来源
- 组合分析就是一种基于参数组合的测试技术，用于发现将参数不同取值作为输入条件时程序的潜在错误
- 判定表方法是使用组合分析的一种技术

判定表的构成

条件桩 列出问题的所有条件	**条件项** 针对所列条件的具体赋值，即每个条件可以取真值和假值
动作桩 列出可能针对问题所采取的操作	**动作项** 列出在条件项（各种取值）组合情况下应该采取的动作

规则：任何一个条件组合的特定取值及其相应要执行的操作，在判定表中贯穿条件项和动作项的一列就是一条规则

判定表的制作

- 列出所有的条件桩和动作桩，作为判定表的行
- 填入条件项，即确定条件桩的可能取值
- 填入动作项，即确定在每列条件桩的取值情况下的动作
- 在初始判定表的基础上简化、合并相似规则或者相同动作

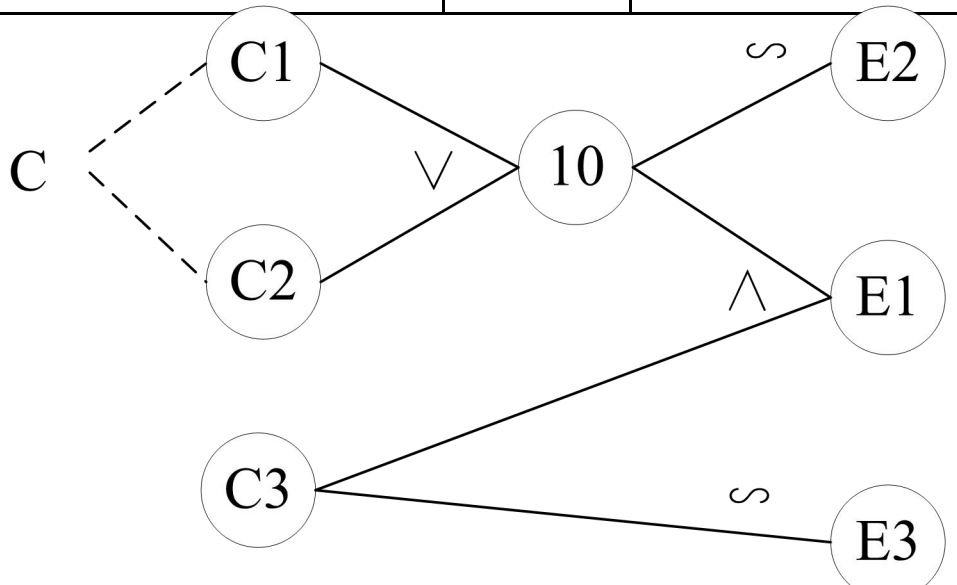
因果图

- 因果图法是另一种使用组合分析的技术
- 该方法借助图形着重分析输入条件的各种组合，每种组合条件就是“因”，它必然有一个输出的结果，即为“果”
- 因果图是一种形式化的图形语言，由自然语言写成的规范转换而成，这种形式的语言实际上是一种使用简化记号表示的逻辑图
  - 使用∧表示“与”、∨表示“或”、∽表示“非”
- 因果图法需要与判定表结合使用，它从软件规格说明书中分析输入与输出数据，关联这些数据，并转换为判定表

- 示例：某软件对用户输入的编码有如下规定：编码的第一个字符必须是A或B，第二个字符必须是一位数字，此情况下给出信息“编码正确”；如果第一个字符不是A或B，则给出信息“编码错误”；如果第二个字符不是数字，则给出信息“修改编码”。

因 与 果

编号	原因	编号	结果
C1	第一个字符是A	E1	编码正确
C2	第一个字符是B	E2	编码错误
C3	第二个字符是数字	E3	修改编码
10	中间原因		



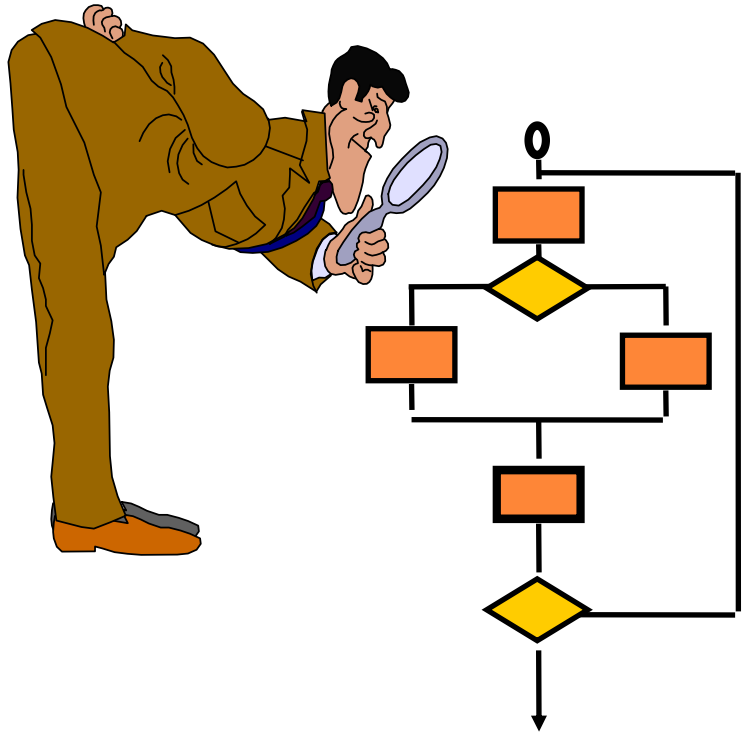
转换为判定表

序号		1	2	3	4/5/6
原因	C1	1	0	0	-
	C2	0	1	0	-
	C3	1	1	1	0
结果	E1	1	1	0	0
	E2	0	0	1	0
	E3	0	0	0	1
测试用例		第一个字符是”A”，第二个字符是数字，给出信息“编码正确”	第一个字符是”B”，第二个字符是数字，给出信息“编码正确”	第一个字符非”A”和”B”，第二个字符是数字，给出信息“编码错误”	第一个字符未定，第二个字符不是数字，给出信息“修改编码”

错误推测法

- 错误推测法又称为探索性测试方法，是测试者根据经验、知识和直觉来推测程序中可能存在的各种错误，从而开展有针对性测试的一种方法
- 主要依赖于测试人员的直觉和经验
  - 优点：测试者能够快速且容易地切入，并能够体会到程序的易用与否
  - 缺点：难以知道测试的覆盖率，可能会遗漏大量未知的软件部分，并且这种测试行为带有主观性且难以复制
- 该方法一般作为辅助手段，即首先采用之前所述的系统化的测试方法，在没有其他方法可用的情况下，再使用错误推测法补充一些额外的测试用例

白盒测试



- 白盒测试也被称为结构测试或逻辑驱动的测试
- 测试人员能够看到程序内部的代码与结构
- 白盒测试的目标是设计出一组测试用例，按照指定的标准覆盖程序中的路径，在运行被测程序时检查程序在顺序、分支与循环流程控制下的运行结果是否符合设计规约的要求

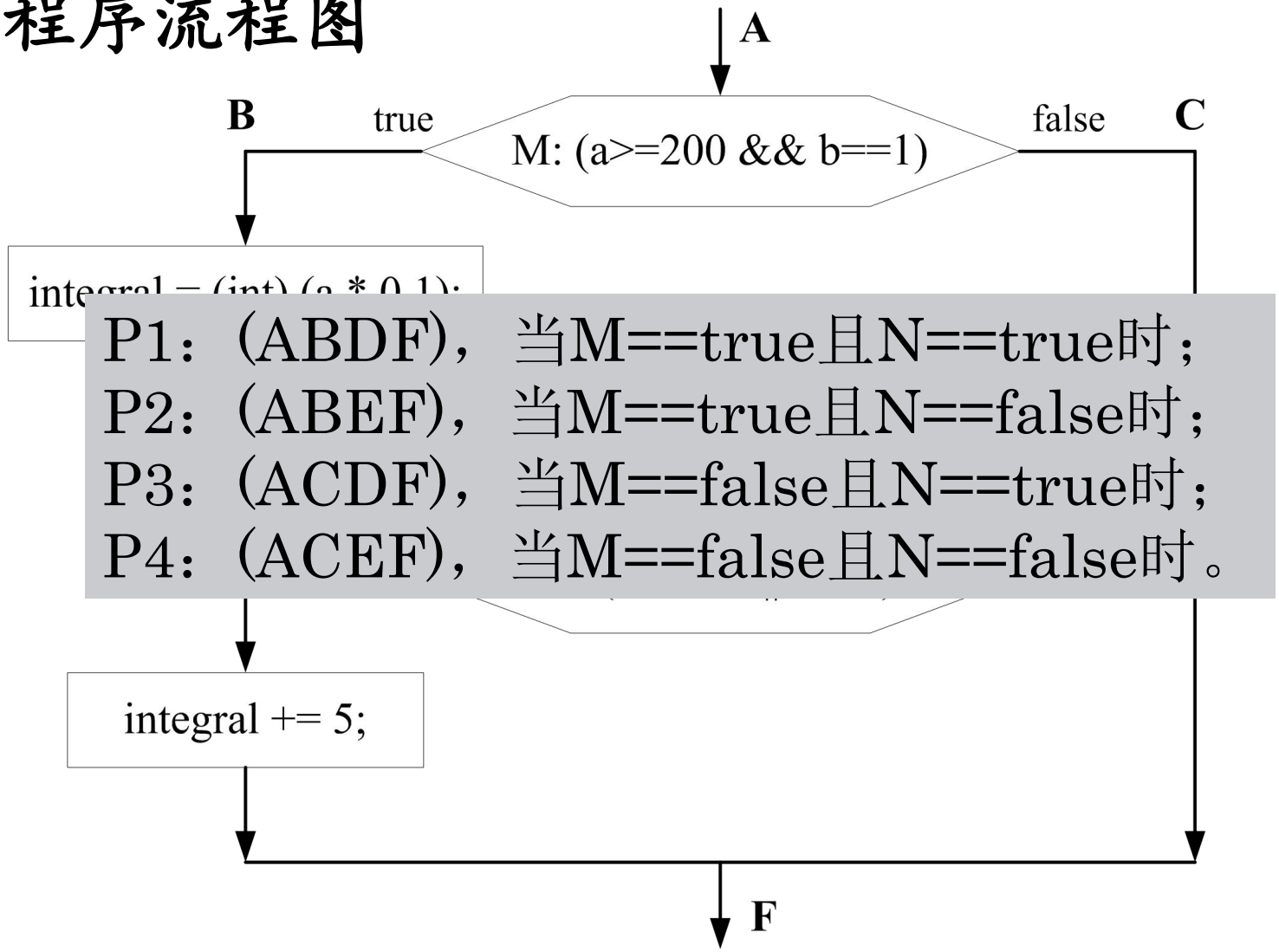
测试对象示例

- 商场促销的积分规则：若购物满200元且用户出示VIP卡，则获取本单10%的积分；若购物满400元或者购物品种大于10件，则另外获赠5个积分。

积分计算功能的代码

```
1.  //a:本单金额；b:VIP卡标志，0代表普通卡，1代表VIP卡；c:购物品种数量
2.  public  int getIntegral(double a, int b, int c) {
3.    int integral = 0;
4.    if (a>=200 && b==1) {
5.      integral = (int) (a * 0.1);
6.    }
7.    if (a>=400 || c>10) {
8.      integral += 5;
9.    }
10.   return integral;
    }
```

- 程序流程图



P1：(ABDF)，当M==true且N==true时；
P2：(ABEF)，当M==true且N==false时；
P3：(ACDF)，当M==false且N==true时；
P4：(ACEF)，当M==false且N==false时。

语句覆盖

- 设计若干测试用例，运行被测程序，使得被测程序中的每条可执行语句至少被执行一次

测试输入	预期输出	判定结果	通过路径
a=350, b=1, c=12	40	M = true, N=true	P1

- 若代码中第3行的&&条件被误写为||条件，则同样能够覆盖所有语句，并且得到正确的预期结果。这是由于语句覆盖是最弱的覆盖标准，它对判定条件中的逻辑错误不进行验证

分支覆盖

- 设计若干测试用例，运行被测程序，使得被测程序中的每个判定的取真分支与取假分支至少被经历一次

测试输入	预期输出	判定结果	通过路径
a=350, b=1, c=12	40	M=true, N=true	P1
a=150, b=0, c=7	0	M=false, N=false	P4

- 若代码中第6行的c>10条件被误写为c>8条件，同样能够得到正确的测试结果

条件覆盖

- 设计若干测试用例，运行被测程序，使得被测程序内每个判定中的每个条件的可能取值至少被满足一次

测试输入	预期输出	条件结果	判定结果	通过路径
a=450, b=0, c=7	5	a >= 200, b <> 1, a>=400, c<=10	M=false, N=true	P3
a=150, b=1, c=12	5	a<200, b==1, a<400, c>10	M=false, N=true	P3

- 这两个测试用例的输入与输出不同，但是他们得到相同的覆盖路径，因此不满足分支覆盖标准

分支/条件覆盖

- 设计若干测试用例，运行被测程序，使得被测程序内每个判定的真值和假值都至少被经历一次，同时要使得每个判定中的每个条件的可能取值至少被满足一次
- 分支/条件覆盖是分支覆盖与条件覆盖的结合体

测试输入	预期输出	条件结果	判定结果	通过路径
a=450, b=1, c=12	50	a>=200, b==1, a>=400, c>10	M=true, N=true	P1
a=150, b=0, c=7	0	a<200, b<>1, a<400, c<=10	M=false, N=false	P4

- 若将代码中第6行的||条件误写为&&条件，也能够获得相同的测试结果

条件组合覆盖

- 设计若干测试用例，运行被测程序，使得被测程序中的每个判定的所有条件组合至少出现一次，且每个判定本身的结果也要至少出现一次

- 8种条件组合
  - ① a>=200, b==1
  - ② a>=200, b<>1
  - ③ a<200, b==1
  - ④ a<200, b<>1
  - ⑤ a>=400, c>10
  - ⑥ a>=400, c<=10
  - ⑦ a<400, c>10
  - ⑧ a<400, c<=10

测试输入	预期输出	条件组合结果	判定结果	通过路径
a=450, b=1, c=12	50	① ⑤	M = true, N=true	P1
a=450, b=0, c=7	5	② ⑥	M = false, N=true	P3
a=150, b=1, c=12	5	③ ⑦	M = false, N=true	P3
a=150, b=0, c=7	0	④ ⑧	M = false, N=false	P4

- 不能保证所有的程序路径都被执行，即P2没有经历过

路径覆盖

- 设计足够的测试用例，用以覆盖程序中的所有可能的执行路径

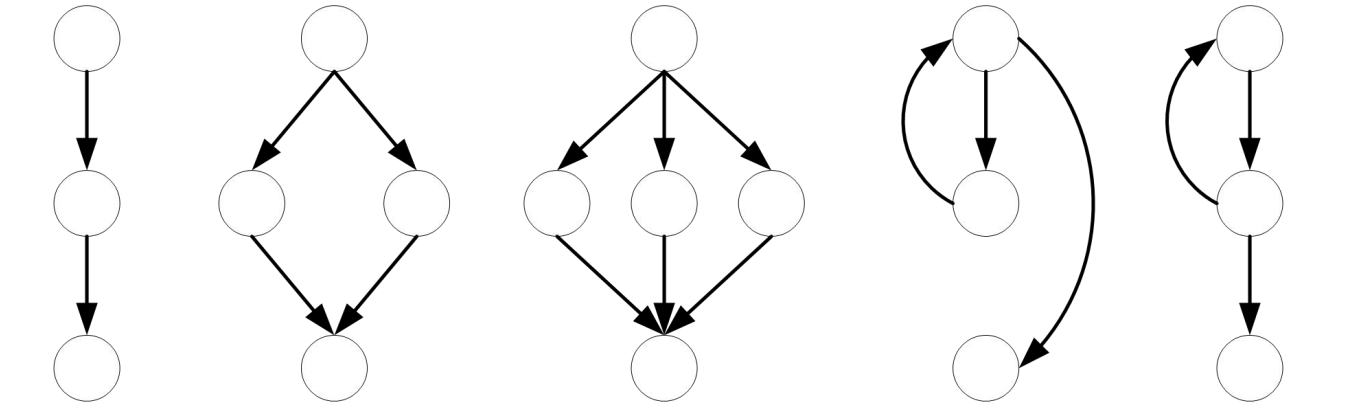
测试输入	预期输出	条件组合结果	判定结果	通过路径
a=450, b=1, c=12	50	① ⑤	M = t r u e , N=true	P1
a=300, b=1, c=7	30	① ⑧	M = t r u e , N=false	P2
a=150, b=1, c=12	5	③ ⑦	M = f a l s e , N=true	P3
a=150, b=0, c=7	0	④ ⑧	M = f a l s e , N=false	P4

- 虽然覆盖了所有的路径，但是却无法覆盖所有的条件组合

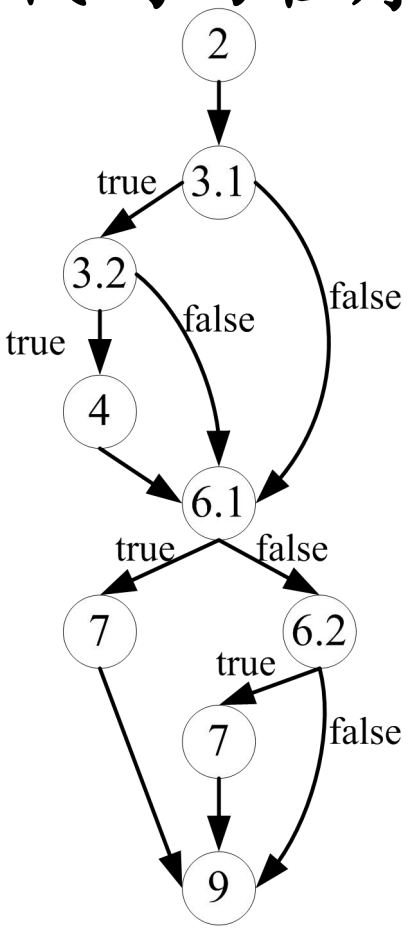
测试输入	预期输出	条件组合结果	判定结果	通过路径
a = 4 5 0 , b=1, c=12	50	① ⑤	M = t r u e , N=true	P1
a = 3 0 0 , b=1, c=7	30	① ⑧	M = t r u e , N=false	P2
a = 1 5 0 , b=1, c=12	5	③ ⑦	M = f a l s e , N=true	P3
a = 1 5 0 , b=0, c=7	0	④ ⑧	M = f a l s e , N=false	P4
a = 4 5 0 , b=0, c=7	5	② ⑥	M = f a l s e , N=true	P3

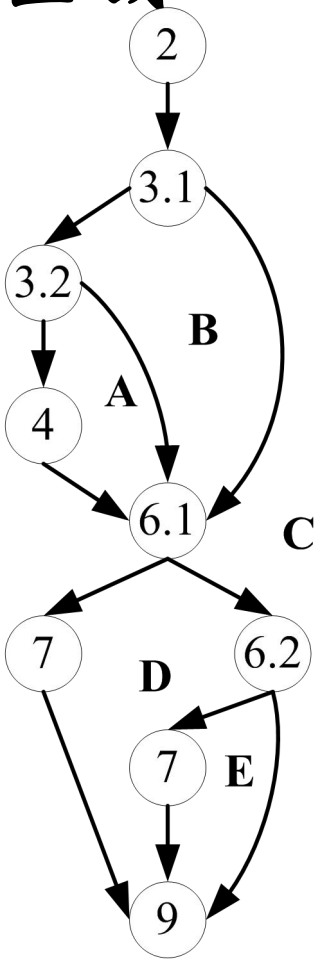
基本路径测试法

- 在程序控制流图的基础上，通过分析控制结构的环路复杂性，导出基本可执行路径集合，从而设计测试用例，这些测试用例能够保证程序中的每条语句至少被执行一次



示例代码的程序控制流图与区域

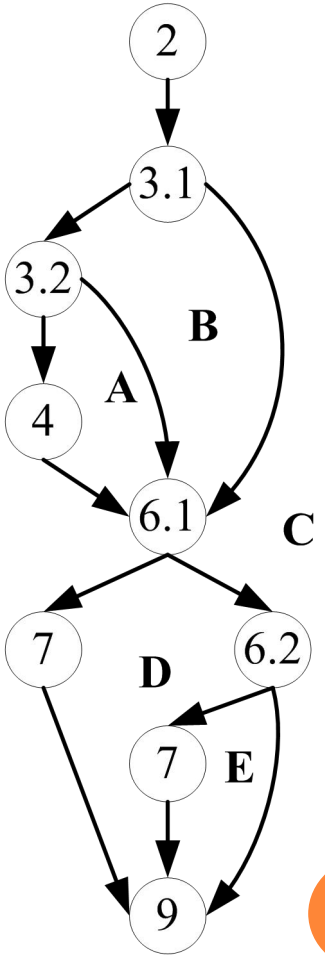




- **计算圈复杂度**
  - 圈复杂度是一种针对程序逻辑复杂性的定量度量，可用于计算程序的基本独立路径数目
- **V(G)**
  - V(G) = 区域数目
  - V(G) = E – N + 2 . E是图中边的数量，N是图中节点的数量
  - V(G) = P + 1 . P是G中分支节点的数量

示例的基本路径

- 路径1：2 – 3.1 – 6.1 – 6.2 – 9
- 路径2：2 – 3.1 – 3.2 – 6.1 – 6.2 – 9
- 路径3：2 – 3.1 – 3.2 – 4 – 6.1 – 6.2 – 9
- 路径4：2 – 3.1 – 6.1 – 6.2 – 7 – 9
- 路径5：2 – 3.1 – 3.2 – 4 – 6.1 – 7 – 9



测试用例

输入	预期输出	路径
a=150, b=1, c=8	0	1
a=250, b=0, c=8	0	2
a=250, b=1, c=8	25	3
a=150, b=1, c=12	5	4
a=450, b=1, c=10	50	5

组合测试

- 在一个具有多参数的软件系统中，输入参数不同取值的组合往往会引起软件的错误
- 测试人员无需完整地测试一个系统的所有参数取值的组合，而仅需对部分参数的所有取值组合进行验证，就能够发现软件的缺陷
- 组合测试技术是一种生成测试用例的技术，能够有效减少测试用例的数目

组合测试的前提

- 将被测软件抽象为一个受到多个因素影响的系统，其中每个因素就是一个系统的输入参数，并且该因素的取值应该是离散且有限的
- 使用组合测试技术得到的每一个测试用例是通过选取每一个因素的其中一个取值构成的

组合测试的组合维度m

- 测试用例要覆盖任意m个因素的所有取值组合
- 二维的组合测试（也称为两因素组合测试）生成的测试用例集就是要覆盖任意两个因素的所有取值组合

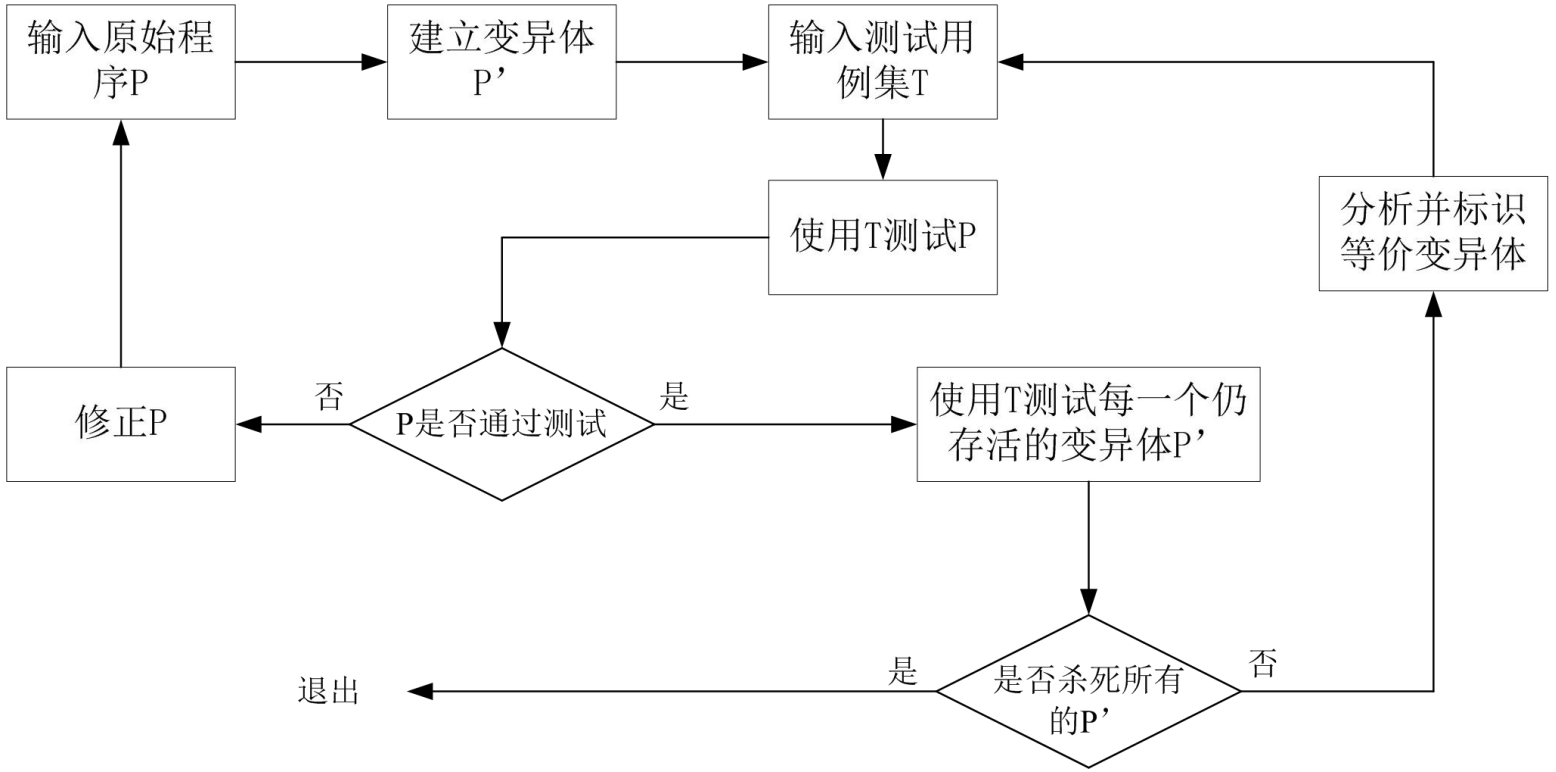
- **组合设计方法**
  - 直接或递归地使用某些代数结构来构造规模较小甚至是最小的N维组合测试用例集
  - 正交表是一种常用的构造组合测试用例集的方法
- **启发式算法**
  - 生成近似最优解的方法
  - 是逐条生成测试用例的过程
  - AETG，TGG，DDA与PICT工具

- **元启发式搜索算法**
  - 个体搜索算法：爬山算法、模拟退火算法、洪水算法、禁忌搜索等
  - 群体搜索算法：遗传算法、蚁群算法等
- **使用元启发式算法生成组合测试用例的策略**
  - 直接搜索，即以不断迭代的方式得到能够满足组合覆盖条件的测试用例集
  - 结合使用启发式算法中的逐条生成测试用例的方式

变异测试

- 软件开发者在编码时可能会引入一些不明显的语法错误
- 目标是对测试用例集的质量进行评估，并提高测试用例集对这些错误的发现能力
- 变异分析充分度（mutation adequacy score）
  - 被测试用例集检测出的故障与所植入的所有故障的数量的比值，它体现了该测试用例集发现缺陷的有效程度

变异测试的基本过程


输入原始程序P
建立变异体P’
输入测试用例集T
使用T测试P
分析并标识等价变异体
修正P
否
P是否通过测试
是
使用T测试每一个仍存活的变异体P’
退出
是
是否杀死所有的P’
否

静态测试

- 是与动态测试相对应的测试技术
- 测试对象是软件系统的文档与代码
- 通过评审的方式对文档内容的正确性、一致性进行验证与确认
  - 组织相关的软件分析、设计与开发人员召开专门的会议，对文档进行阅读，并对有疑问的部分进行讨论。一旦发现文档中存在缺陷，应该尽早对文档的内容进行纠正，并同时修改与该文档相关的其他文档或代码

- **针对代码的静态测试（代码评审）**
  - 通过阅读代码从而检查代码和设计的一致性，代码对标准以及编程规范的符合性，代码的清晰性、可读性，代码的逻辑与结构的正确性，代码的安全性等
- **桌面检查**
  - 由程序员检查自己编写的程序
- **走查**
  - 相对比较正式的代码评审过程
  - 成立走查小组
  - 预先将与代码相关的材料分发给走查小组内每个成员
  - 为待测程序准备一批具有代表性的测试用例
  - 让与会者充当“计算机”的角色，用他们的头脑来执行测试用例，在纸上或黑板上随时记录程序状态与执行路径

审查

- 正式的检查和评估方法
- 由审查小组通过阅读、讨论与争议，对程序进行静态分析的过程
- 缺陷检查表将程序中可能发生的各种错误进行分类，对每一类列举出尽可能多的典型错误
- 在审查会议上，首先由程序员逐句讲解程序的逻辑。在此过程中，程序员或其他小组成员可以提出问题，展开讨论，根据缺陷检查表审查错误是否存在。程序员在讲解过程中可能发现许多原来自己没有发现的错误，而讨论和争议则促进了问题的暴露。审查过程中发现的问题应被快速记录，并在会后再被具体解决。

- 请总结各种测试技术的目的、覆盖标准与注意点。

软件测试类型

单元测试

- 单元测试针对一个程序单元，验证其各方面的软件特性
- 单元模块的接口测试
- 局部数据结构的测试
- 单元执行路径的测试
- 错误处理的测试
- 单元边界条件的测试

单元模块的接口测试

- **测试接口正确与否应该考虑以下的因素：**
  - 调用本单元时的实际参数与形式参数在个数、类型等方面是否匹配、一致；
  - 本单元调用其他单元时，在实际参数与形式参数在个数、类型等方面是否匹配、一致；
  - 调用预定义函数或库函数时，在使用的参数个数、属性和次序方面是否正确；
  - 是否修改了输入型（只读）参数的值；
  - 全局变量的定义在各个单元中是否一致；
  - 是否把某些约束作为参数传递。

- **如果被测单元通过外部设备进行输入/输出操作时，还应该考虑以下7个因素：**
  - 文件属性是否正确；
  - OPEN语句与CLOSE语句是否正确；
  - 规定的I/O格式说明与I/O语句是否匹配；
  - 缓冲区大小与记录长度是否匹配；
  - 在文件被使用前是否已经打开了文件；
  - 在结束文件处理时是否已经关闭了文件；
  - 是否处理了输入输出或者文字性的错误。

局部数据结构的测试

- **数据结构方面的缺陷：**
  - 不正确或不一致的类型说明；
  - 使用尚未被初始化赋值的变量；
  - 错误的初始值或错误的默认值；
  - 变量名拼写错误；
  - 变量的数值上溢、下溢或者地址错误。

单元执行路径的测试

- **程序单元中常见的计算错误包括：**
  - 不正确的或误用的运算优先级次序；
  - 运算的对象在类型上不兼容；
  - 变量的初值错误；
  - 运算精度不够；
  - 表达式的符号错误。
- **控制流错误包括：**
  - 不同数据类型对象之间进行比较；
  - 不正确的逻辑运算符或优先级；
  - 因浮点数运算精度问题而造成的两值比较不相等；
  - 关系表达式中不正确的变量和比较符；
  - 循环未按照预期的次数，多一次或少一次循环；
  - 错误的或不可能的循环终止条件；
  - 迭代发散时不能退出；
  - 错误地修改了循环变量。

错误处理的测试

- **错误处理方面的问题：**
  - 单元输出的出错信息难以理解；
  - 显示的错误与实际的错误不相符；
  - 对错误条件的处理不正确；
  - 在程序自定义的出错处理部分运行之前，系统已进行干预；
  - 错误陈述不足以对错误进行定位。

单元边界条件的测试

- 软件经常在边界上出现错误
- 边界包括输入数据范围的边界，以及程序执行控制流的边界（例如，最后一次循环）等

集成测试

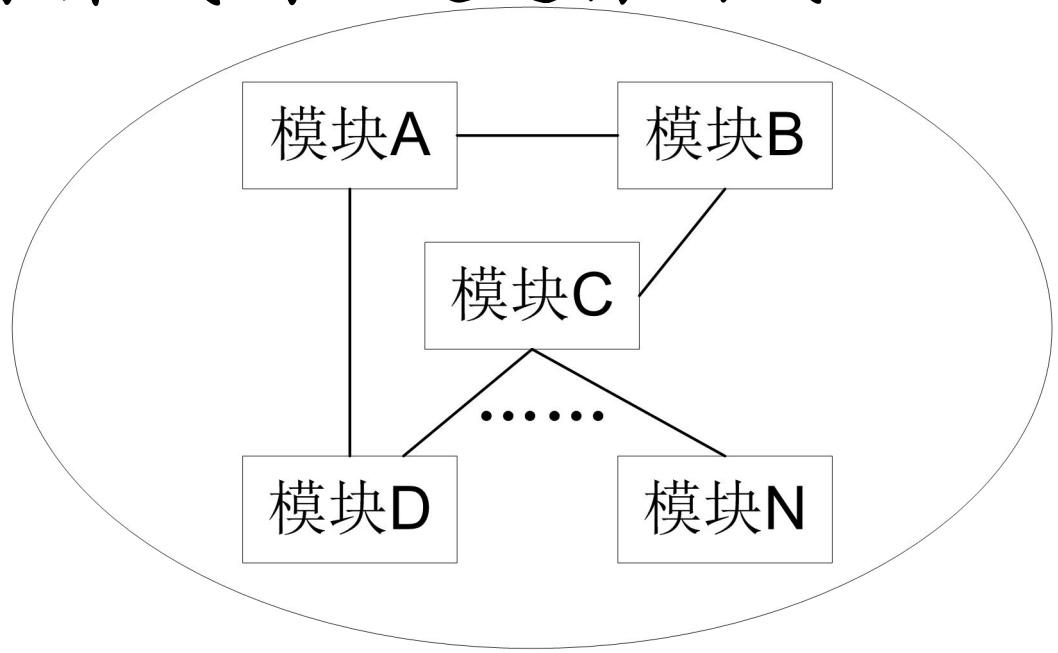
- **集成测试，也被称为组装测试或联合测试**
  - 将已分别通过测试的单元按照设计要求组合起来再次进行的测试，其主要目的在于检查这些程序单元之间的接口是否存在问题
- **导致错误的因素包括**
  - 不同模块对参数或值存在不一致的解释。
  - 模块交互后进行计算的误差累计达到了不能接受的程度，或者模块的接口参数取值超出值域或者容量。
  - 全局数据结构出现错误
  - 当模块使用那些未在接口中明确提到的资源
  - 当规格说明不完整且描述不准确

集成测试策略

- 大爆炸式集成测试策略
- 自顶向下集成测试策略
- 自底向上集成测试策略
- 三明治集成测试策略
- 其他集成测试策略

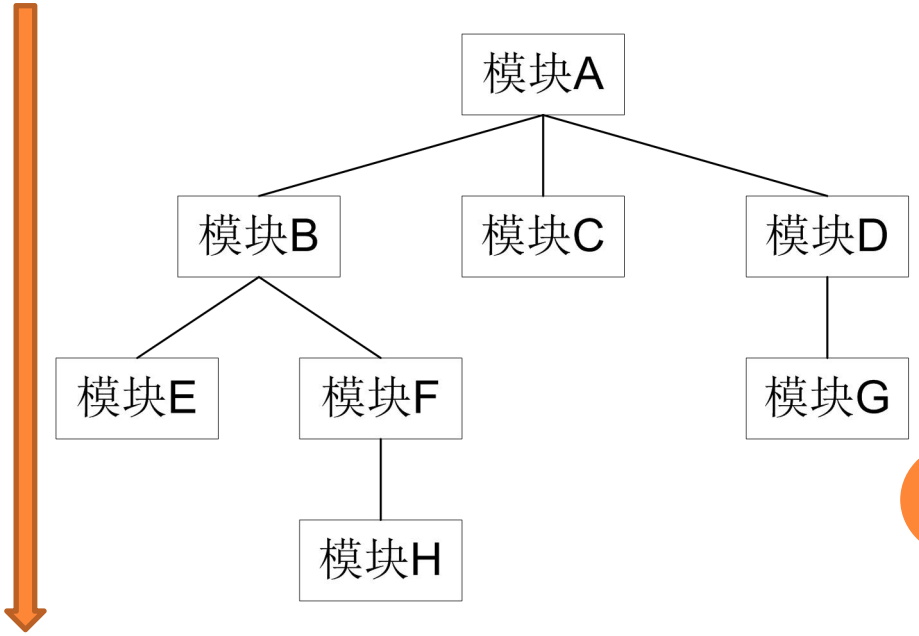
大爆炸式集成测试策略

- 将所有通过单元测试的模块一次性地按照设计要求集成到一起进行测试



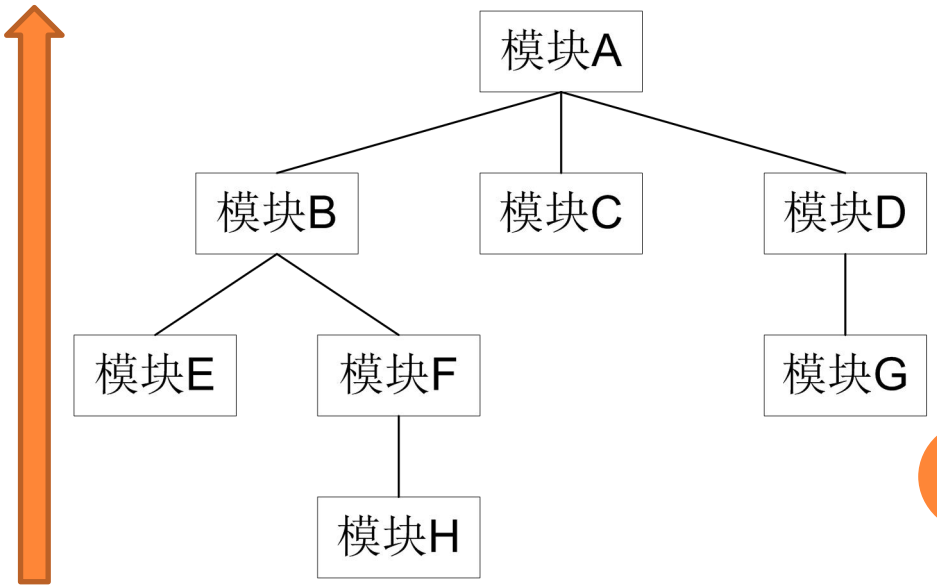
自顶向下集成测试策略

- 按照系统概要设计所包含的层次结构图，以主程序模块为中心，自上而下按照深度优先或者广度优先策略，对各个模块一边组装一边进行测试
- 深度优先
  - A-B-E-F-H-C-D-G
- 广度优先
  - A-B-C-D-E-F-G-H



自底向上集成测试策略

- 从系统概要设计所包含的层次结构图的最底层模块开始进行组装和集成的方式



三明治集成测试策略

- **混合渐增式的测试策略，它综合了自顶向下和自底向上两种集成方法的优点**
  - 在模块层次结构图中确定以哪一层为界来使用三明治集成策略。
  - 对该层下面的各层使用自底向上的集成策略。
  - 对该层上面的层次使用自顶向下的集成策略。
  - 把处于该层的各模块同相应的下层集成。
  - 对系统进行整体测试。

其他集成测试策略

- 基于调用图的集成
- 基于功能的集成
- 基于风险的集成
- 基于事件的集成

系统测试

- **系统测试**
  - 将整个软件系统看作一个整体进行测试，包括对功能、质量（如性能、安全性等），以及软件所运行的软硬件环境等各方面进行整体的测试
- **由黑盒测试工程师在整个系统集成完毕以后进行测试**
  - 前期主要测试系统的功能是否满足需求，即进行所谓的功能测试
  - 后期主要测试系统运行过程中的质量属性是否满足用户期望的要求，即执行所谓的非功能性测试
  - 在系统测试中还需要测试系统在不同的软硬件环境中的兼容性等

功能测试

- **检查实际软件的功能是否符合用户的需求**
  - 根据产品的需求规格说明书和测试需求列表，验证产品的功能实现是否符合产品的需求规格
- **主要的测试目标涵盖界面、数据、操作、逻辑、接口等几个方面**
  - 程序安装、启动正常，有相应的提示框、适当的错误提示等；
  - 每项功能符合实际要求；
  - 系统的界面清晰、美观；菜单、按钮操作正常、灵活，能处理一些异常操作；
  - 能接受正确的数据输入，对异常数据的输入可以进行提示、容错处理等；
  - 数据的输出结果准确，格式清晰，可以保存和读取；
  - 功能逻辑清楚，符合使用者习惯；
  - 系统的各种状态按照业务流程而变化，并保持稳定；
  - 支持各种应用的环境，能配合多种硬件周边设备，与外部应用系统的接口有效；
  - 软件升级后，能继续支持旧版本的数据

非功能性测试

- 关注软件系统在运行中所体现出的质量属性，检查软件系统是否能够满足在需求说明书中所规定的质量
- 测试种类
  - 性能测试、压力测试、容量测试、安全性测试、可靠性测试与容错性测试等

负载测试技术

- **软件系统的性能指标**
  - 响应时间
  - 并发用户数
  - 吞吐量
  - 系统占用资源
- **负载测试技术**
  - 通过不断加载系统负载，例如逐渐增加模拟用户的数量或其它加载方式，来观察不同负载下系统的响应时间、数据吞吐量、系统占用的资源的变化情况。

对性能测试工具的需求

- **性能测试工具使用两种策略自动生成并发用户访问，提供两种不同的负载类型**
  - Flat：一次性地加载所有的用户，然后在预定的时间段内保持这些用户的持续运行
  - Ramp-up：用户是交错上升的，即工具每隔几秒增加一些新的用户
- **性能测试工具需要依赖测试人员所编制的测试脚本才能完成其任务**
- **在测试执行过程中，工具记录相关的性能指标**

性能测试

- 通过测试以确定软件系统运行时的性能表现的一种测试方式
- 性能测试的目的可以概括为：在真实环境下检测系统性能，评估系统性能以及服务等级的满足情况，同时分析系统瓶颈、优化系统

性能测试的类型

- 基准性能测试
- 性能规划测试
- 渗入测试
- 峰谷测试

压力测试

- 模拟实际应用的软硬件环境及用户使用过程的系统负荷，长时间或超大负荷地运行测试软件，来测试被测系统的性能、可靠性、稳定性等质量属性
- 基于负载测试技术

压力测试 vs. 性能测试

- 性能测试的主要关注点是模拟多种正常、峰值以及异常负载条件来对系统的各项性能指标进行测试
- 压力测试则是通过确定一个系统的瓶颈或者不能接收的性能点，来获得系统能提供的最大服务级别的测试

压力测试类型

- 并发性能测试
- 疲劳强度测试
- 大数据量测试

容量测试

- 通过测试预先分析出反映软件系统应用特征的某项指标的极限值
- 确定测试对象在给定时间内能够持续处理的最大负载或工作量
- 基于负载测试技术开展

安全性测试

- **检查系统对非法侵入的防范能力**
- **测试对象：**
  - 物理环境的安全（物理层安全）
  - 操作系统的安全性（系统层安全）
  - 网络的安全性（网络层安全）
  - 应用的安全性（应用层安全）
  - 管理的安全性（管理层安全）

可靠性测试

- 产品在规定的条件下和规定的时间内完成规定功能的能力，它的概率度量称为可靠度
- 软件可靠性与软件缺陷有关，也与系统输入和系统使用有关
- 可靠性测试是通过测试来度量软件可靠度的过程
  - 通过错误发现率DDP（Defect Detection Percentage）来展现
  - DDP等于测试发现的错误数量除以已知的全部错误数量
  - 在测试中查找出来的错误越多，实际应用中出错的机会就越小，软件也就越成熟，其可靠程度也就越高。

容错性测试

- **检查软件在异常条件下是否具有防护性的措施或者某种灾难性恢复的手段**
- **可通过两种方式进行**
  - 对系统输入异常数据或进行异常操作，以检验系统是否具有自保护性，即是否能够自我处理错误。
  - 通过各种手段，让软件强制性地发生故障，然后验证系统已保存的用户数据是否丢失、系统和数据是否能尽快恢复。

可用性测试

- 让一群有代表性的用户对产品进行典型操作，同时测试人员或开发人员在一旁观察、记录
- 用户界面测试，即UI测试
  - 着重于验证用户与计算机进行交互时的正确性与合理性

兼容性测试

- 验证软件系统与其所处的上下文环境的兼容情况
- 针对硬件兼容性、浏览器兼容性、数据库兼容性、操作系统兼容性等方面

安装与卸载测试

- **安装测试的目标是找到软件产品在安装阶段所引发的问题**
  - 对环境变量的检测和解释
  - 文件复制时的错误
  - 系统和环境配置出错
  - 软件和硬件不兼容
  - 后台噪声（例如病毒检查程序）
- **卸载测试的目的是验证能够成功卸载软件产品的能力**

恢复测试

- 证实在克服硬件故障（包括掉电、硬件或网络出错等）后，系统能否正常地继续工作，并不对系统造成任何损害
- 可采用各种人工干预的手段，模拟硬件故障，故意造成软件出错，并对此进行检查

指标/协议测试

- 测试依据是我国已经发布的信息技术产品强制性标准、推荐性标准等相关国家标准或行业标准
- 数据内容标准测试
- 通信协议标准测试
- 字符集和代码页测试

本地化测试

- 保证本地化的软件与源语言软件具有相同的功能和性能
- 保证本地化的软件在语言、文化、传统观念等方面符合当地用户的习惯
- 尽可能多地发现软件中由于本地化而引起的bug

验收测试

- **检查软件系统是否已完成了用户所提出的需求**
  - 验收测试站在用户的角度对软件进行检查
- **α 测试**
  - 软件开发公司组织用户或内部人员模拟各类用户对即将面市的软件产品（称为α版本）进行测试，试图发现错误并对其进行修正。
  - 在公司内部搭建的与实际应用相类似的软件运行环境中执行
  - 通过α测试或经过修改后的软件产品被称为β版本

β 测试

- 软件开发公司组织各方面的典型用户在日常工作中实际使用β版本，并要求用户报告异常情况、提出批评意见
- 基于这些反馈，软件开发公司再对β版本进行修正与完善
- 当β测试完成后，软件产品即可被正式发布

回归测试

- 当软件系统发生变更后，就应当执行回归测试
- 目的是确保新引入的变更不会影响已有软件系统的行为，尤其是那些未发生变化的软件部分的行为
- 提高回归测试的有效性，降低回归测试的成本
  - 测试用例的精简
  - 测试用例的筛选
  - 测试用例的优先级排序

测试用例的精简

- 从原有测试用例集中识别并移除废弃的或冗余的测试用例的过程
  - 废弃的测试用例是指已不再适用于验证系统特性的测试用例
  - 冗余的测试用例是指在不断累积测试资产的过程中所出现的多个具有相同的输入和输出的测试用例
- 测试用例的精简技术旨在从原始的测试用例集中找出能够符合系统的所有测试需求的一个测试用例子集
- 缩减是永久性的
- 期望找到一个测试用例数目最小的子集

测试用例的筛选

- 在执行一次回归测试时，选择那些与程序被修改部分相关的测试用例来对变更后的程序再次进行测试
- 筛选是一个临时的过程，它所选择的测试用例子集仅对特定的回归测试有效
- 在其他的回归测试中，可能会筛选出不同的子集

测试用例的优先级排序

- 找出测试用例集中测试用例的最理想化的排列顺序，当在回归测试过程中按照顺序执行这些测试用例时，能够最大化测试的效益
- 测试用例的优先级排序技术都基于一个排序准则，该准则指明了如何对测试用例的特性进行评估，从而根据该特性进行排序
  - 基于覆盖率的排序准则

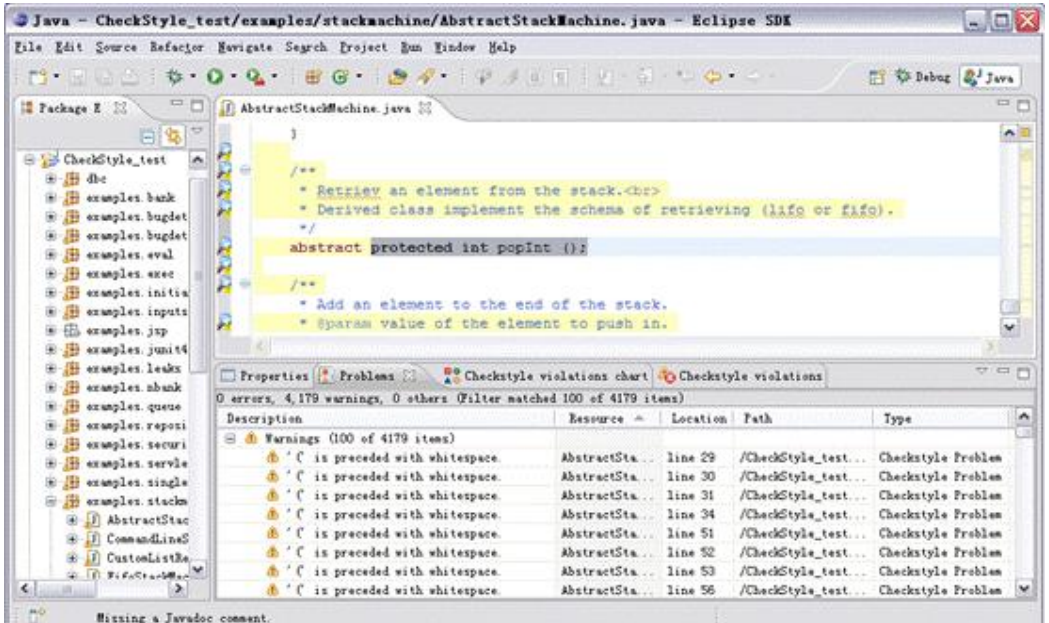
软件测试工具

静态代码分析工具

- 在代码构建过程中帮助开发人员快速、有效地定位代码缺陷并及时纠正这些问题
- 基于坚实的理论基础，使用到缺陷模式匹配、类型推断、模型检查、数据流分析等自动化技术手段，对代码进行完整的扫描，检查被测程序的语法、结构、过程、接口等来验证程序的正确性，找出代码隐藏的错误和缺陷
  - 参数不匹配，有歧义的嵌套语句，错误的递归，非法计算，可能出现的空指针引用

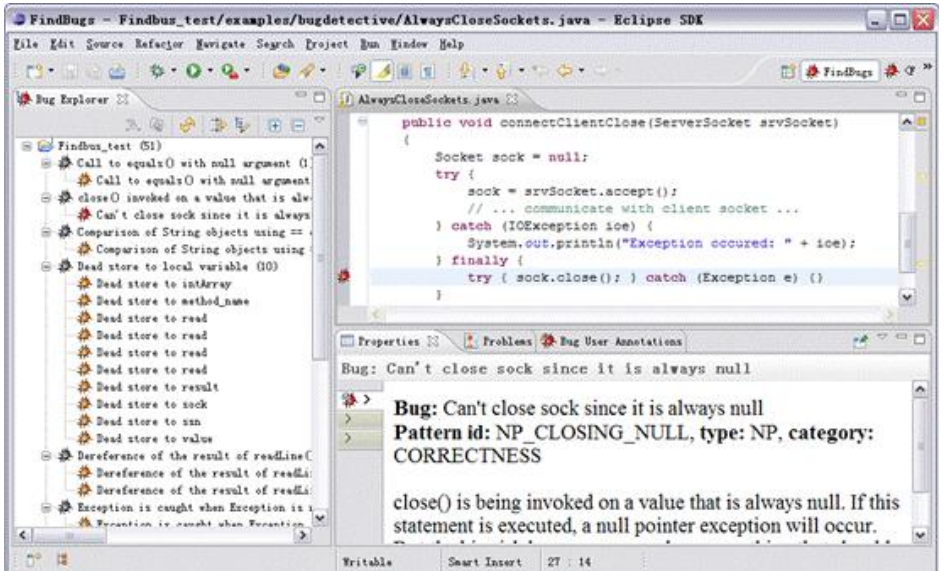
CHECKSTYLE

- 对代码编码格式、命名约定、Javadoc、类设计等方面进行代码规范和风格的检查
- 支持用户根据需求自定义代码检查规范



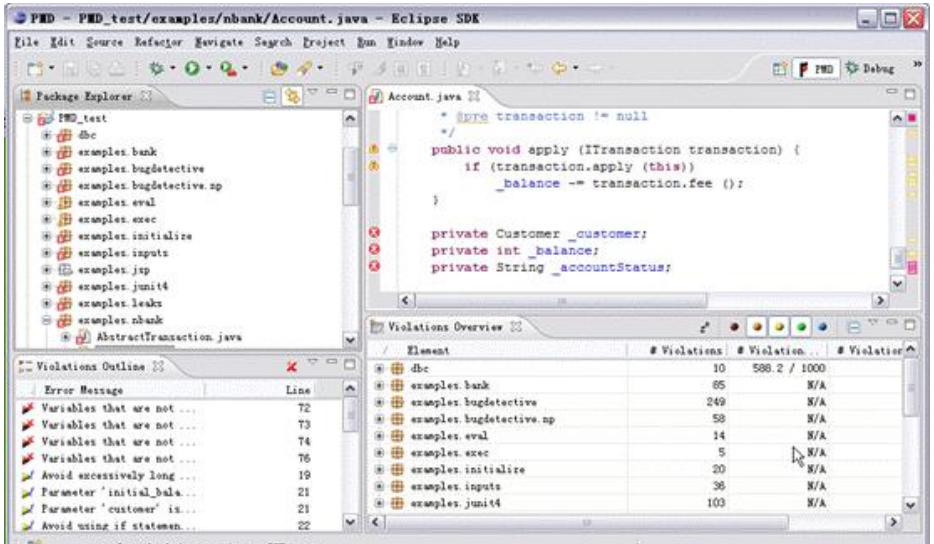
FindBugs

- 通过检查类文件或Jar文件，将字节码与一组缺陷模式进行对比来发现代码缺陷
- 为用户提供定制bug模式的功能，用户可以根据需求自定义FindBugs的代码检查条件



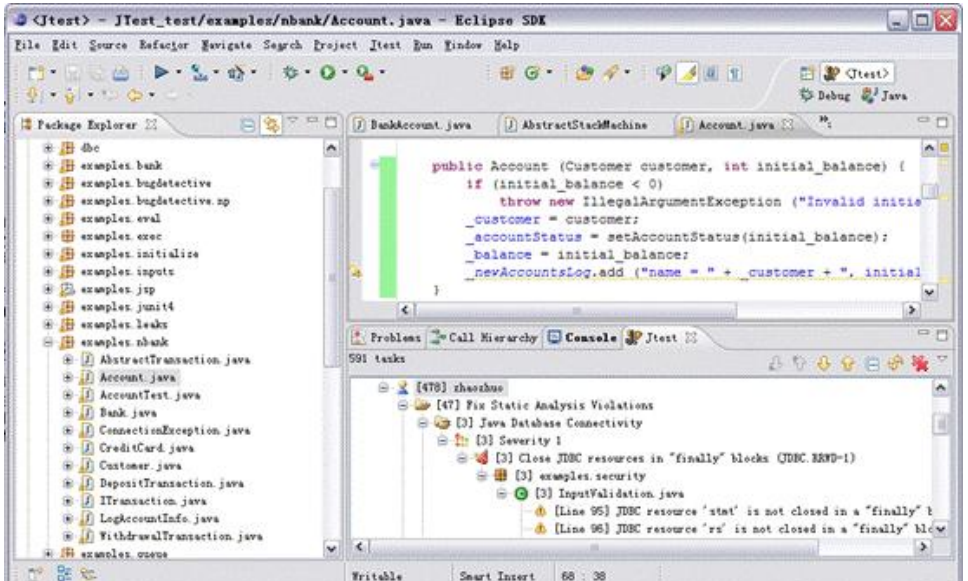
PMD

- 通过其内置的编码规则对Java代码进行静态检查，主要包括对潜在的bug、未使用的代码、重复的代码、循环体、创建新对象等问题的检验
- 支持开发人员对代码检查规范进行自定义配置



JTEST

- 按照其内置的超过800条的Java编码规范自动检查并纠正这些隐蔽的且难以修复的编码错误
- 支持用户自定义编码规则和代码检查配置



系统测试工具

- Selenium
  - 用于Web功能测试的开源工具
- QTP
  - 商用的功能测试工具
- Jmeter
  - 用于负载测试的开源工具
- LoadRunner
  - 商用的负载测试工具

SELENIUM

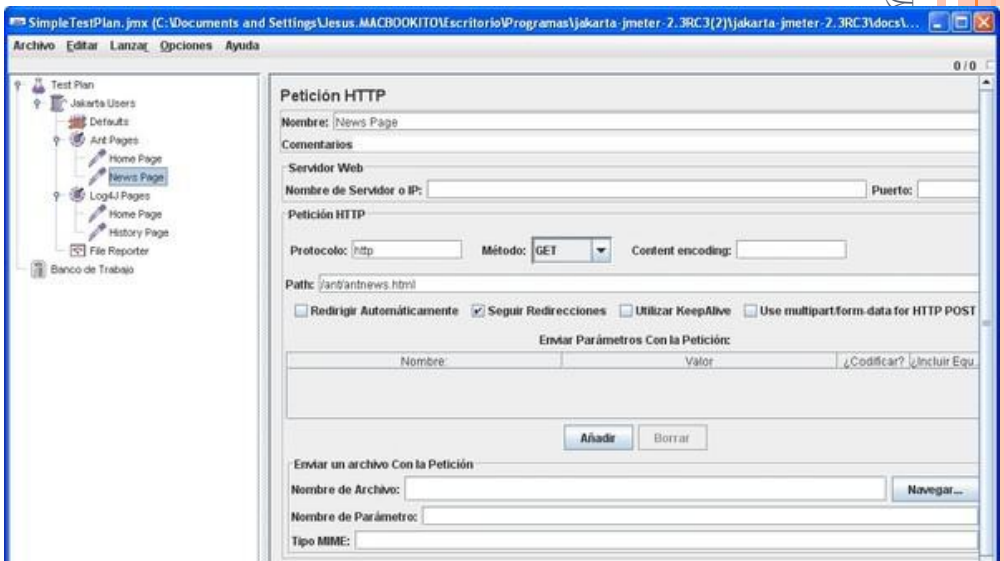
- 开源的基于Web的测试工具
- 采用JavaScript单元测试工具JSUnit为核心，模拟真实的用户操作
- 能够和浏览器进行通信，把测试用例的数据发送给浏览器执行，从而达到自动测试的目的
- Selenium 包括如下不同的产品：
  - Selenium Core
  - Selenium IDE
  - Selenium RC

QTP

- HP公司研制的、目前主流的自动化测试工具
- 支持广泛的平台和开发语言，包括Web、VB、.NET、Java等
- 执行重复的手工测试，降低测试的成本
- 步骤
  - 录制测试脚本
  - 编辑测试脚本（加入检查点）
  - 调式测试脚本
  - 运行测试脚本
  - 分析测试结果

JMeter

- 模拟对服务器、网络或对象的大量负载，即在同一时刻或一段时间内自动生成大量的并发用户或线程，请求系统的服务，从而在不同的负载压力下测试系统的强度，验证、分析系统的整体性能
- 配合使用脚本录制工具Badboy为其生成测试脚本



LoadRunner

- 预测系统行为和性能的工业标准级负载测试工具，适用于各种体系结构的自动负载，预测系统行为并优化系统性能
- 通过模拟多用户实施并发负载验证系统的整体性能与容量
- 使用实时性能监测的方式来确认和查找运行时系统的问题

- **使用LoadRunner进行负载测试的步骤：**
  - 计划负载测试（性能测试或压力测试）。
  - 创建用户脚本（VU脚本）
  - 定义、设计场景
  - 运行场景
  - 分析结果

面向对象软件的测试

面向对象软件测试的难点

- **封装为测试带来的难点**
  - 如果类未提供足够的存取函数来表明对象的实现方式和内部状态，那么测试也就难以验证运行时的对象实例所执行的功能是否正确
  - 单元测试的粒度应该提高为类级别或对象级别
- **继承为测试带来的难点**
  - 一个类内部隐含地包含了它所继承的父类的属性或方法，或者对继承方法稍作修改进行覆盖
  - 一个基类能够被一个派生类所替换
  - 父类的测试用例也能够被一定程度地复用于子类的测试

多态为测试带来的难点

- 可能产生未预想的场景，即某些绑定能正确地工作但并不能保证所有的绑定都能正确地运行
- 多态绑定的对象可能很容易地将消息发送给错误的类，执行错误的功能，还可能导致一些与消息序列和状态相关的错误

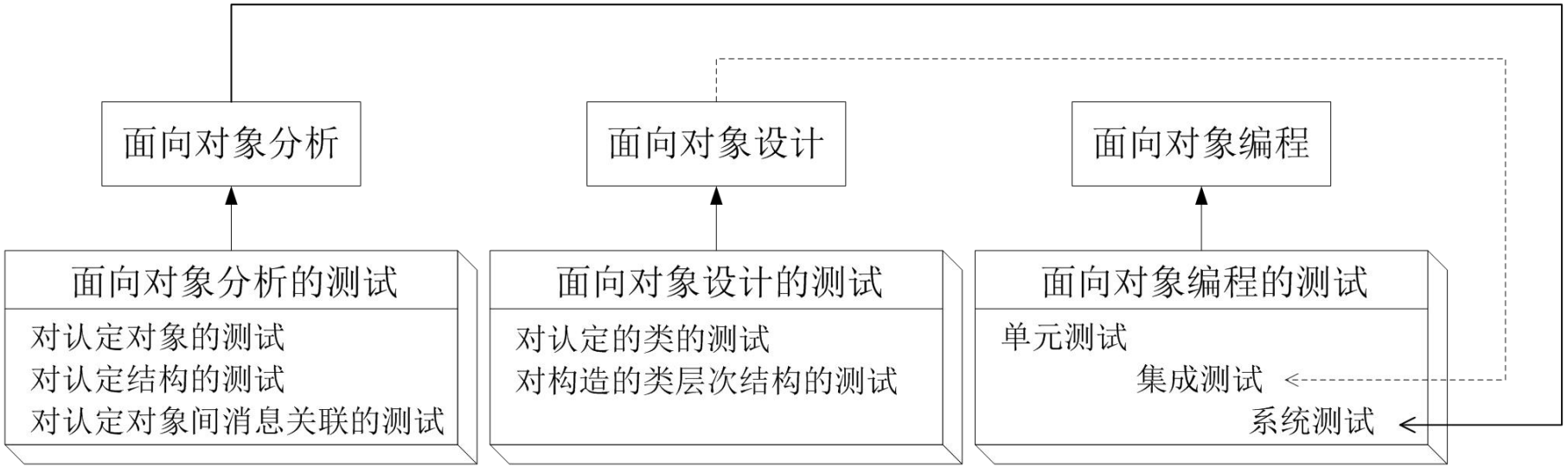
对象状态转换为测试带来的难点

- 不能仅仅检查一个方法在特定输入数据情况下产生的输出结果是否与预期相吻合，还要考虑对象的状态变化

对象消息交互为集成测试带来的难点

- 一个面向对象程序的完整功能实际上是由一组消息连接起来的方法序列，方法可以位于不同的类中，因此为了实现特定的软件功能，需要激活或调用属于不同类或对象的多个成员方法，从而将方法集成起来形成一个调用链
- 传统集成测试的自顶向下或自底向上的集成策略将不适用

面向对象软件的测试方法



面向对象分析的测试

- **对认定对象的测试**
  - 验证所认定的对象是否全面、对象的属性是否描述正确、对象的操作是否全面、同一对象的实例是否具有与其他实例不同的共同属性等
- **对认定结构的测试**
  - 验证系统中对象的继承层次或分解层次是否正确
  - 针对分类结构的测试
  - 针对组装结构的测试
- **对认定对象间消息关联的测试**
  - 验证对象之间是否存在合理的、可行的消息交互
  - 关注于消息的可达性、消息应是同步还是异步、消息发送对象的多态性等

面向对象设计的测试

- **对认定的类的测试**
  - 验证系统的设计类是否包含了与其关联的对象的所有公共属性与操作，并且这些设计类之间是否具有显而易见的区别
- **对构造的类层次结构的测试**
  - 类层次接口是否能够涵盖所有定义的类
  - 是否能够体现面向对象分析中所定义的实例关联与消息关联
  - 子类是否具有父类不包括的新特性
  - 子类间的共同特性是否完全在父类中得以体现

面向对象编程的测试

- **面向对象的单元测试**
  - 验证类中所有操作的正确性，以及类在实例化之后在实际运行环境中状态变化的正确性
  - 白盒测试方法与黑盒测试方法
  - 针对一个类的所有有意义方法的测试用例
  - 在类级别上针对类的状态进行测试
  - 1）基于状态的测试
  - 2）基于响应状态的测试

面向对象的集成测试

- **基于线程的集成**
  - 根据一个线程的从头至尾的执行过程，集成所涉及的一组类，这组类之间的方法调用能够满足该线程提供正确的功能
  - 应对每一个线程都进行集成测试，以保证系统主要交互流程的正确性
- **基于使用的测试**
  - 首先测试那些几乎不依赖于其他类的类，这些类被称为独立类。在独立类被测试完成后，集成那些使用独立类的类，这些类被称为依靠类。按照依赖的层次序列不断进行集成，直至构造出完整的系统。

面向对象的系统测试

- 验证系统在实际运行时能够满足用户需求，在测试时应尽量搭建与用户实际使用相同的测试环境
- 面向对象的系统测试可采用传统的系统测试方法
- 功能测试
- 非功能性测试

- 请总结面向对象软件测试的主要难点。

高级软件工程

软件测试

The End