软件测试技术

- 从软件质量说起
- 软件测试基础
- 测试用例设计
- 软件测试策略

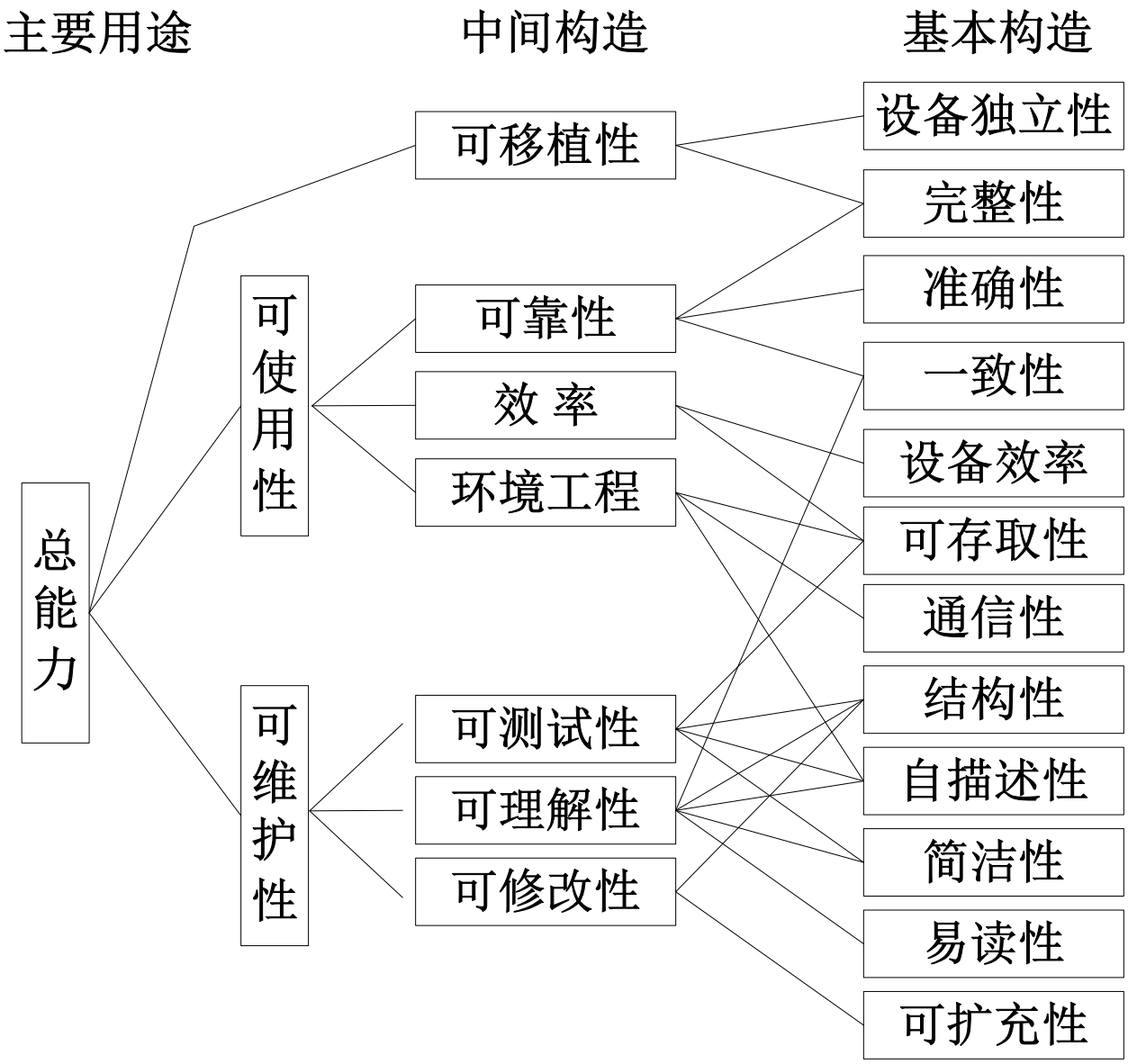
软件测试技术

- 从软件质量说起
- 软件测试基础
- 测试用例设计
- 软件测试策略

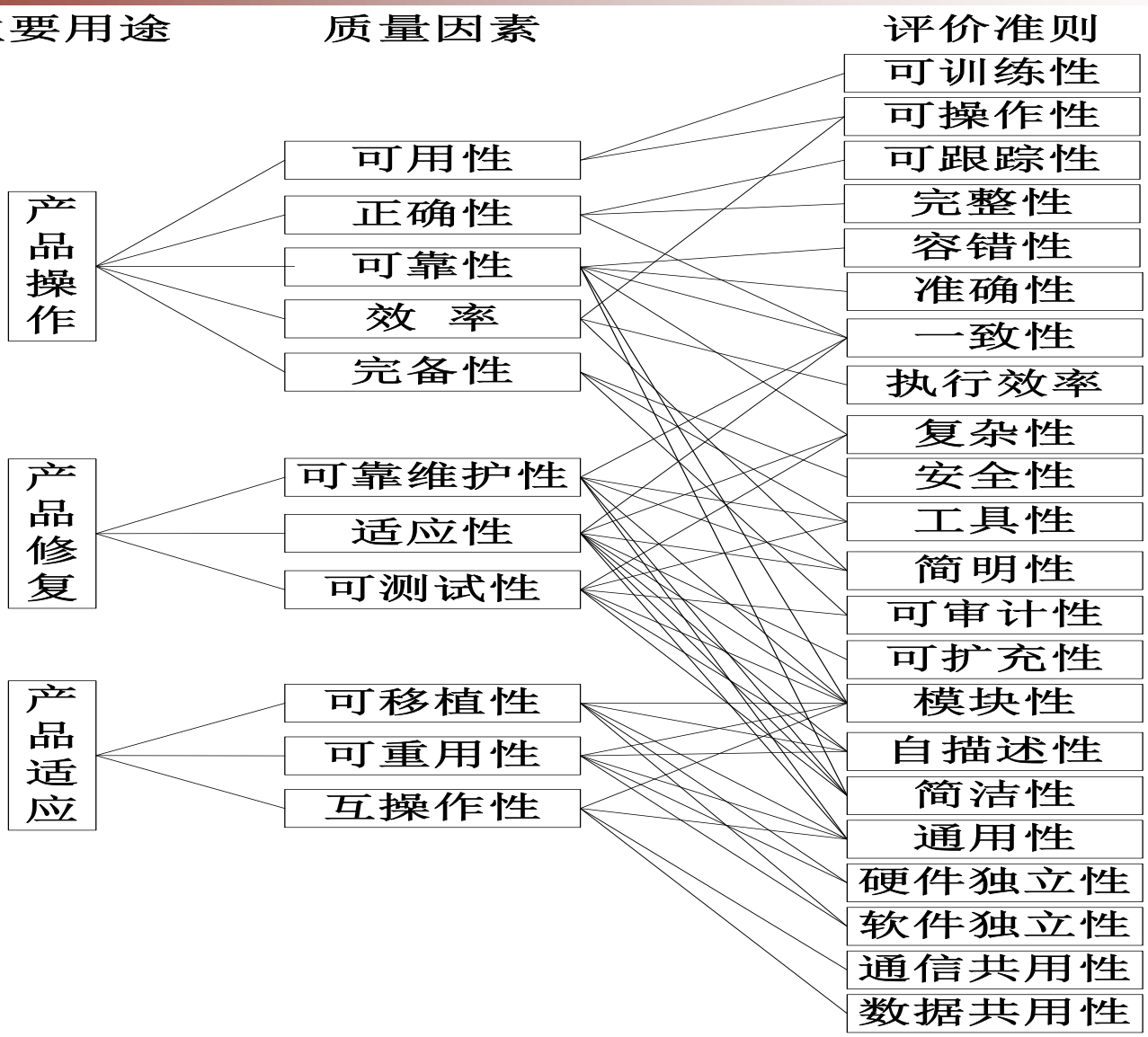
软件质量

- 如何评价一个软件产品的好坏
- 软件质量
  - 与明确确定的功能和性能需求的一致性
  - 与明确成文的开发标准的一致性
  - 与所有专业开发的软件所期望的隐含的特性的一致性

质量模型—Boehm模型



质量模型-McCall模型



如何保证软件质量

软件质量保证(SQA, Software Quality Assurance)是一项有计划的、系统的和规范性活动，包括：

- 选用合适的技术方法：过程、方法、工具
- 实施正式技术评审(FTR)
- 变更控制和管理
- 多层次的软件测试
- ……

软件测试技术

- 从软件质量说起
- 软件测试基础
- 测试用例设计
- 软件测试策略

软件测试(Software Testing)

- 是一种提高软件质量的有效手段
- 是为发现错误而执行程序的过程
  - 根据软件开发各阶段的规格说明和内部结构而精心设计一批测试用例（即输入数据及其预期的输出结果）并利用这些测试用例去运行程序，以发现程序错误的过程
- **IEEE**
  - **Testing is the process of exercising or evaluating a system or system component by manual or automated means to verify that it satisfies specified requirements**

测试目标

- 测试目标
  - 测试是一个为了寻找错误而运行程序的过程
  - 一个好的测试用例是指很可能找到迄今为止尚未发现的错误的用例
  - 一个成功的测试是指发现了至今未发现的错误的测试

测试目的

- 测试目的
  - 验证软件实现了规格说明（需求、设计）的要求
  - 发现程序中的错误/缺陷
  - 确定是否可以使用、可交付
  - 了解性能的限制
  - 了解系统不能做什么
  - 评估系统的能力和质量
  - 验证文档
  - ……

几个测试相关的概念

- 验证和确认
- 测试和调试
- 缺陷和错误
- 可测试性

验证和确认（V&V）

- 验证(Verification)
  - 证明在软件生存期各个阶段之间的逻辑协调性、完备性和正确性
  - 保证软件正确地实现了某一特定功能的一系列活动
  - 保证各阶段的交付的产品满足其输入规格说明
  - “我们是否正确地完成了产品？”
- 确认(Validation)
  - 证实在一个给定的外部环境中软件的逻辑正确性。
  - 保证软件的实现满足了用户需求的一系列活动
  - 保证最终产品满足用户需求
  - “我们是否完成了正确的产品？”

测试和调试

- 调试(debugging)
  - 在测试发现一个错误后消除错误的过程
  - 编码阶段末期的排错活动

测试	调试
已知条件，用户预定义的过程，有预期结果	不知道初始条件
需要计划、设计、安排日程	没有限制
展示是错或对	推理性过程
不需要排错	要进行排错
可以没有设计的知识	不能没有设计知识
可以由外部的人进行	必须由内部的人进行

关于软件缺陷或错误

有几个与软件缺陷相关的术语

- **Error(错误): 指编写错误的代码，包括语法错误和逻辑错误；**
- **Bug(小错误、缺陷、不足、过失 …)：一般指在计算机程序中存在的错误(error)、缺陷(flaw)、疏忽(mistake)或者故障(fault)，这些bug使程序无法正确的运行；**
- **Defect(缺陷): 软件与需求不一致，常指软件无法正确完成需求所要求的功能，也称之为bug；**
- **Fault(故障): 存在于组件、设备或子系统中异常的条件或者缺陷，常常会导致系统的失败；**
- **Failure(失效): 一个组件、设备、子系统或系统无法(不具备相应的能力)去完成它设计的任务；**

可测试性(Testability)

- 可测试性
  - 是一种软件质量特性，表达了对软件进行测试时信息获取的难易程度，包括：
    - ✓可控性：　便于对软件的内部状态进行控制
    - ✓可观测性：能够对软件的内部状态进行观测
- 可分两类可测试性
  - 需求可测试性
  - 软件可测试性

基本测试原理

- 测试应尽早开始
  - 缺陷放大原理：在生命周期前一阶段引入的缺陷，如果不及时排除，将导致更多的缺陷。
- 制定测试计划
  - 测试工作真正开始前制定完善的测试计划；
  - 在需求分析阶段针对确认/系统测试任务开始编制测试计划，之后分别在概要、详细设计阶段补充集成、单元测试内容。从程序构造阶段逐步完成单元、集成、确认/系统、验收测试任务；
  - 严格执行测试计划，避免测试的随意性。

基本测试原理(续)

- 设计测试用例
  - 测试用例必须包括有效的和期望的输入条件以及无效的和不期望的输入条件；
  - 测试用例应定义所期望的输出和结果。
- 管理测试结果
  - 对每个测试结果做全面调查和分析；
  - 管理测试计划、测试用例、测试结果，为维护提供方便。
- 测试独立于开发

  开发人员往往不愿意否定自己的工作，其思考常常有一种思维定向。
- 穷举测试是不可能的

妨碍有效测试的各个方面

- 很多方面的因素会阻碍软件测试的效果
  - 过于乐观、过于相信系统
  - 自负、对有效测试的否定态度
  - 不希望失败
  - 测试人员和开发人员之间的冲突
  - 非结构化的测试
  - 测试成本
  - 交付时间的限制
  - ……

软件测试技术

- 从软件质量说起
- 软件测试基础
- 测试用例设计
- 软件测试策略

测试用例(Test Case)

- 测试用例
  - 为证实程序符合/不符合要求的程序一次执行
  - 展示软件实现是否符合预期的要求
- 测试用例的主要内容
  - 初始状态、输入数据、预期输出、测试过程
  - 实际结果、结果评价
- 设计测试用例的两种技术
  - 白盒测试、黑盒测试

实例：测试用例

功能模块编号	HR_ORG_001	功能模块名	组织结构数据批量保存和批量查询
测试目的	验证批量保存以及批量查询功能		
测试步骤	1、批量录入组织机构信息，员工信息，岗位信息并保存到 oJadb 数据库中。 2、通过批量查询校验批量保存数据的正确性。		
预期结果	组织结构数据被正确地保存到 oJadb 数据库中，并可以被正确地查询到。		
实际结果	批量保存正常，批量查询正常		

实例：测试用例

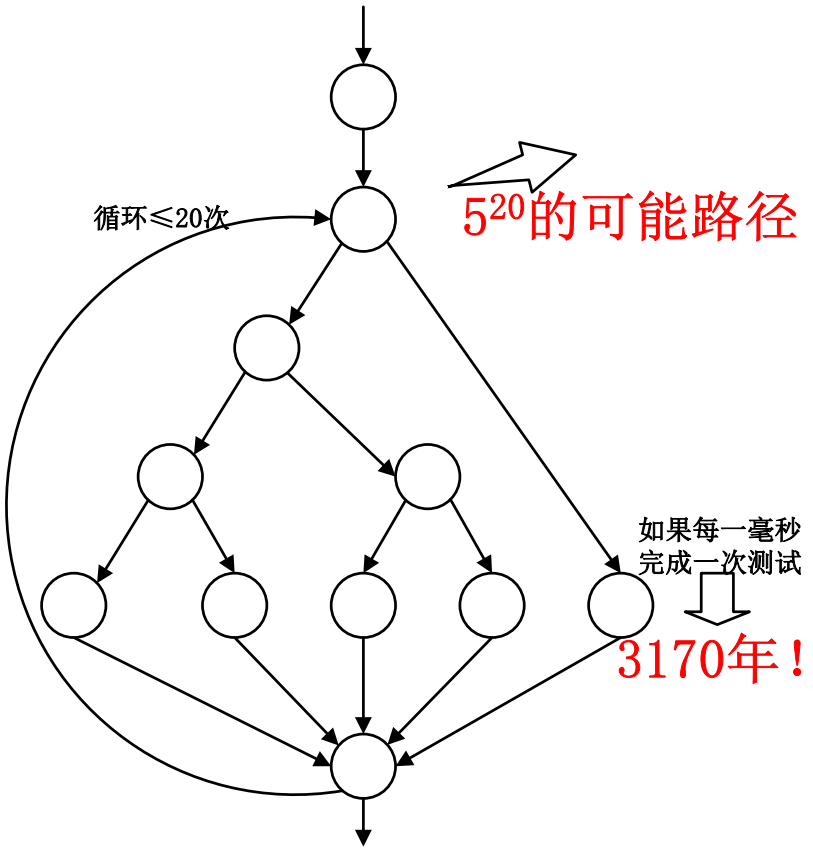
内容	测试过程	输入	输出
注册新用户	创建一个新用户	输入符合要求的用户信息	用户创建成功
		输入不符合要求错误的用户信息	用户创建失败
删除用户	删除一个用户	删除	删除成功
登录系统	使用一个用户登录系统	使用不存在的用户登录	登录不成功，提示用户不存在
		使用存在的用户登录，但密码错误	登录不成功，提示密码错误
修改用户信息	选择一个用户，对其信息进行修改，将修改后的信息保存到数据库	输入正确格式的用户信息	修改成功，并保存到数据库
		输入错误格式的用户信息	修改失败，没有保存到数据库

设计测试用例的方法

- 测试用例设计的典型方法
  - 利用生产数据进行测试
  - 采用某种测试用例选择技术

穷举测试是不可能的

- 白盒测试
  - 如果每1ms完成一次测试，则需3170年
- 黑盒测试
  - 假定：输入X、Y；输出Z均为整数，在32位计算机上，输入数据的组合为 $2^{32}X2^{32}=2^{64}$(种)
  - 如程序每执行一次需时1ms，那么把所有数据测试一遍，则需5亿年



什么是好的测试用例

- 好的测试用例
  - 发现错误的可能性很高
  - 不冗余
    - ✓构造一个与其他测试用途完全相同的测试
  - 最佳品种
    - ✓使用最可能找到所有错误的测试
  - 既不会太简单，也不会太复杂

设计测试用例的基本技术

- 设计测试用例的两种基本技术
  - 白盒测试
    - ✓基于实现的测试
    - ✓基于代码的测试
  - 黑盒测试
    - ✓基于需求测试
    - ✓基于规格说明的测试

白盒测试

- 白盒测试

  一种测试用例设计的技术，利用作为构件层设计的一部分而描述的控制结构来生成测试用例。

- 利用白盒测试设计测试用例，可达到如下目标
  - 保证一个模块中所有独立路径至少被执行一次
  - 对所有的逻辑值均需测试真和假
  - 在上下边界及可操作的范围内执行所有的循环
  - 检测内部数据结构以确保有效性

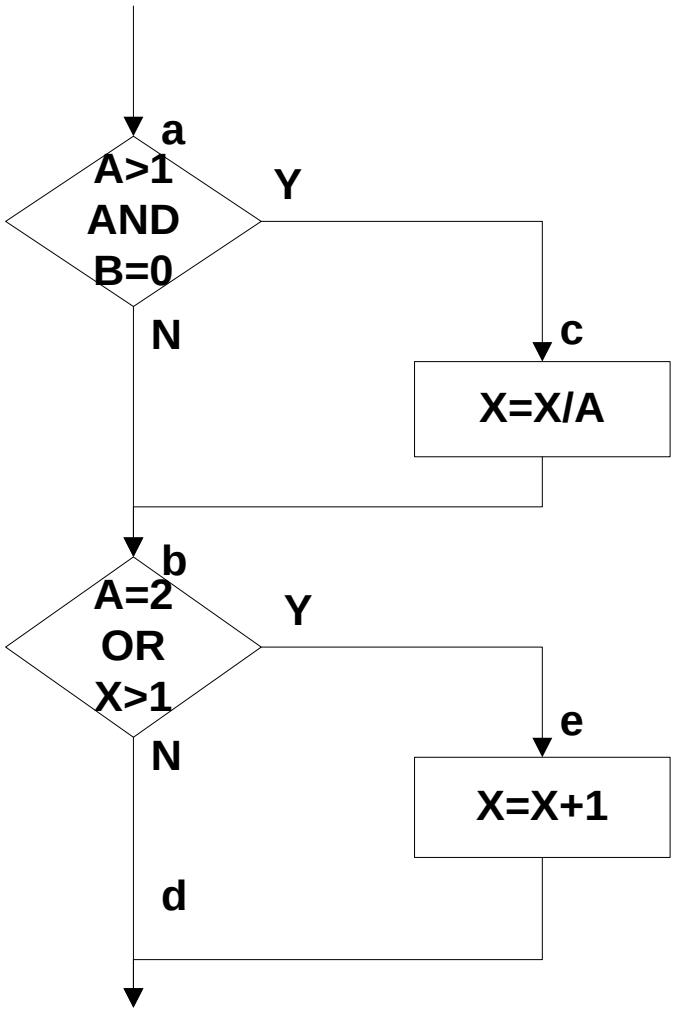
利用逻辑覆盖进行白盒测试

- 逻辑覆盖

  以程序内部逻辑为基础的测试技术，考虑的是程序内部逻辑被覆盖的程度。

- 按覆盖程度不同，逻辑覆盖分为：
  - 语句覆盖：每条语句至少执行一次
  - 分支覆盖：每个分支都至少通过一次
  - 条件覆盖：分支中各个条件的所有可能都至少执行一次
  - 分支/条件覆盖：分支中的每个条件的所有结果至少出现一次，同时使各分支本身的所有可能也至少出现一次
  - 多重覆盖：使各分支中条件的所有可能组合至少出现一次
  - 循环覆盖：用来检查循环构造的有效性
  - 路径覆盖：用来检查所有路径的执行情况

实例：逻辑覆盖



- 语句覆盖：设计一条能通过路径ace的测试用例：A=2，B=0，X=3

- 分支覆盖：能通过路径acd和abe；可选择以下两组数据：
  - A=3，B=0，X=1(acd)
  - A=2，B=1，X=3(abe)

- 条件覆盖：使a、b的每种可能都执行一次；可选择以下两组数据：
  - A=2,B=0,X=4(ace)
  - A=1,B=1,X=1(abd)

白盒测试总结

- 测试用例包括
  - 初始状态、输入数据、预期输出、测试过程
  - 实际结果、结果评价
- 完全测试是不可能的
- 测试覆盖可用来指导测试者设计测试用例，同时也可以用来检测测试的进度
- 测试覆盖主要有：
  - 语句覆盖
  - 分支覆盖
  - 条件覆盖
  - 分支/条件覆盖
  - 多重覆盖
  - 循环、路径覆盖

黑盒测试

- 黑盒测试
  - 根据模块的功能需求设计测试用例
- 检查目标
  - 是否有不正确或遗漏了的功能
  - 在接口上，输入能否正确地输入和输出
  - 是否存在数据结构错误或外部信息访问错误
  - 性能能否满足要求
  - 是否有初始化和终止性错误

黑盒测试方法

- 基于输入域
  - 等价类划分、边界值分析
  - 域测试
- 基于图的测试
  - 因果图
  - 判定表、功能图
- 其他测试
  - 错误推测法、变异测试、比较法测试

等价类划分

- 等价类

  指某个输入域的子集合，在该子集合中，各个输入数据对于揭露程序中的错误是等效的。

  - 有效等价类

    对于程序的规格说明来说，是合理的，有意义的输入数据构成的集合。

  - 无效等价类

    对于程序的规格说明来说，是不合理的，无意义的输入数据构成的集合。

示例：有效等价类和无效等价类

输入值	有效等价类	无效等价类
区间[a,b]	a≤x≤b	X<a, x>b
特定值a	a	X<a, x>a
布尔量	true	false
一组值(a,b,c)	a,b,c	X不属于{a,b,c}
要求符合某规则	符合规则	不符合情况1 不符合情况2 …

等价类测试步骤

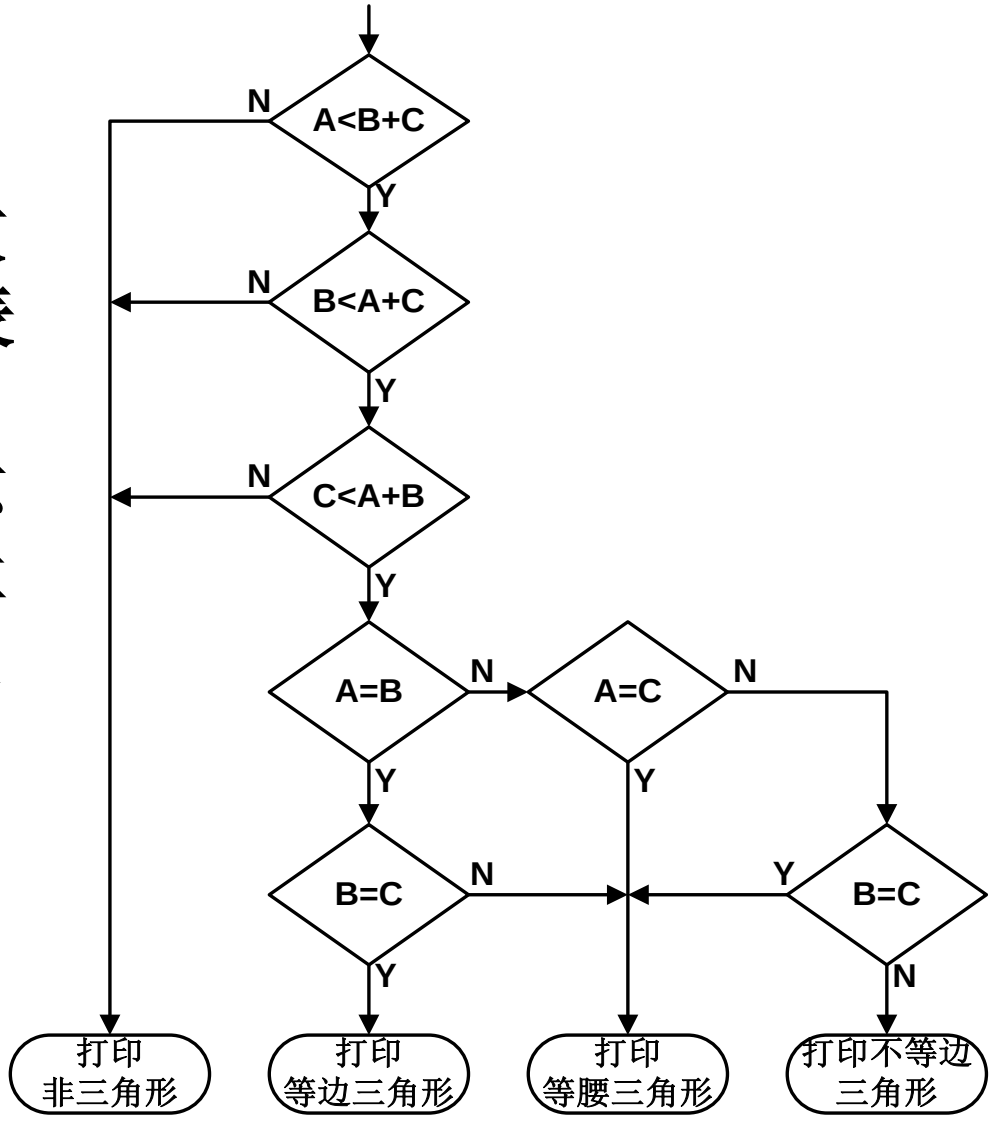
- 识别每个功能单元的参数
  - 参数
  - 可影响功能单元操作的环境对象
    - ✓系统参数、外部数据对象、外部设备状态等
- 识别每个参数的特性
  - 类型、长度、状态值等
- 将每个特性类划分为不同的等价类
  - 有效等价类和无效等价类

等价类测试（续）

- 确定各个等价类之间的限制，如
  - 在一个等价类和另一个等价类中值不能同时存在
  - 一个值必须比另一个值大等
- 设计测试用例
  - 为每个等价类规定一个唯一的编号
  - 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，重复到所有的有效等价类都被覆盖为止。
  - 设计一个新的测试用例，使其仅覆盖一个尚未被覆盖的无效等价类，重复到所有的无效等价类都被覆盖为止。

实例：三角形分类程序测试设计

- 三角形分类程序
  - 程序读入三个整型数据值(A、B、C)，表示三角形的三个边。按照等边、等腰、不等边和非三角形的要求，把它们分类输出



合理的输入：有效等价类

类型	输入			输出
	A B C	A B C	A B C	
等 边	5, 5, 6			等边
等 腰	5, 5, 6	5, 6, 5	6, 5, 5	等腰
不等边	3, 4, 5	4, 5, 3	5, 4, 3	不等边
非三角形	3, 3, 7	3, 7, 3	7, 3, 3	非三角形

不合理的输入：无效等价类

类型	输入			输出
	A B C	A B C	A B C	
“0”数据	0, 3, 4	3, 0, 4	4, 3, 0	不是上述四类三角形
负数据	-3, 4, 5	4, 5, -3	5, -3, 4	
丢失数据	-, 4, 5	4, -, 5	5, 4, -	
非数据	x, 4, 5	4, 5, x	5, x, 4	

边界值分析

- 经验表明
  - 使用正好等于、小于或大于边界值的数据，对程序进行测试，发现错误的概率比较大
- 设计测试用例时，应选择一些边界值，即边界值分析
  - 例：三角形程序中，如果使用等价划分，至少可以找出两个等价类：一个是三角形的有效等价类；一个两数之和不大于第三个数的无效等价类

    A=3,B=4,C=5；A=1,B=2,C=4
  - 但是，却忽视了极易出现的错误：如把**A+B>C**错误地写成**A+B≥C**，上述两组测试数据均无法发现。可见边界值分析的必要性
  - **A=1,B=2,C=3** 这组数据可以发现上述错误

边界值分析技巧

- 如果输入/输出条件代表一a和b为边界的范围，测试用例应当包括a、b、略大于a和略小于b的值。
- 如果输入/输出规定了值的个数，测试用例应当包括最大个数、最小个数、比最小个数多一个、比最大个数少一个的测试数据。
- 如果程序的规格说明给出的输入或输出域是有序集合，则应选择集合的第一个元素和最后一个元素作为测试用例。
- 如果程序中使用了一个内部数据结构，则应当选择这个内部数据结构的边界上的值做为测试用例。

边界值分析与等价类划分

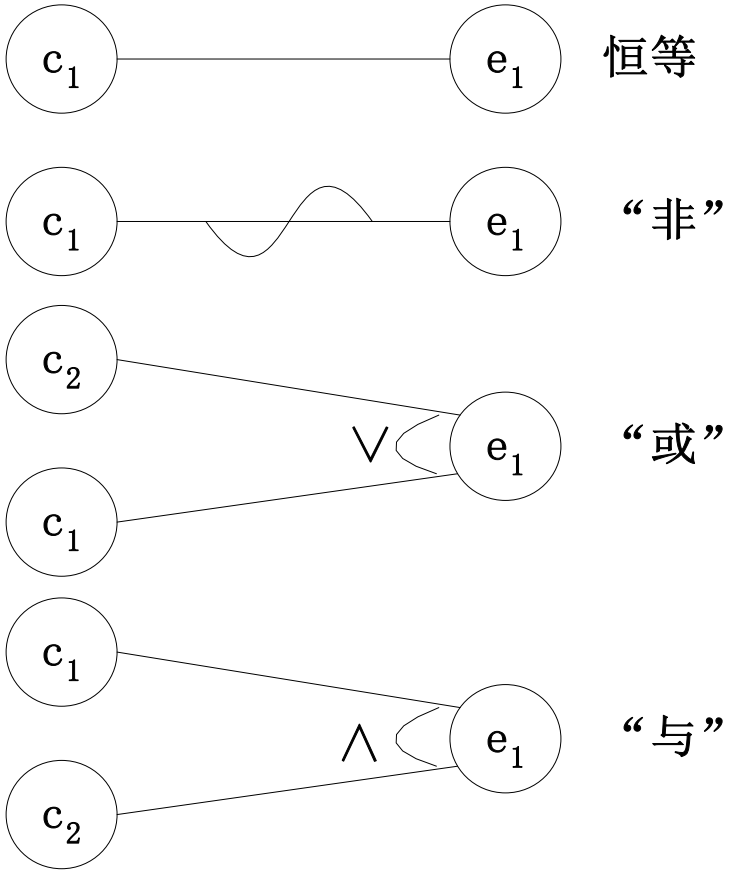
- 边界值分析着重检查等价类边界上和边界附近的错误
- 实际工作应用中

  常把逻辑覆盖、等价划分和边界值分析测试结合使用，即把白盒和黑盒测试技术结合起来，不仅可检查设计的内部要求，又可以检查设计的接口要求。

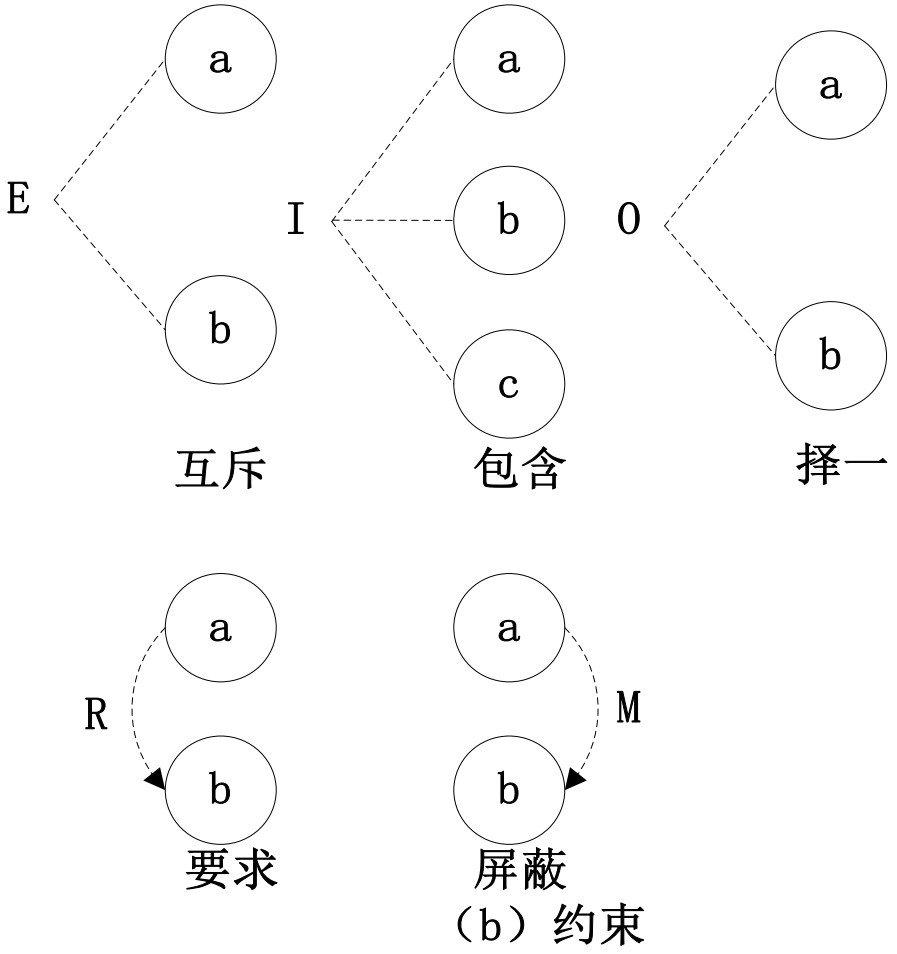
因果图

- 因果图(cause-effect diagrams)
  - 利用条件与对应动作之间的逻辑关系，着重检查各输入条件的组合
    - ✓如：两个输入值的乘积超过了存储的限制，程序发生错误，等价划分和边界值分析，都不能发现这类错误
    - ✓它们均未考虑输入情况的各种组合
  - 因果图的基本原理是通过画因果图把用自然语言描述的功能说明转换为判定表最后为判定表的每一列设计一个测试用例

因果图中的主要符号



（a）符号



（b）约束

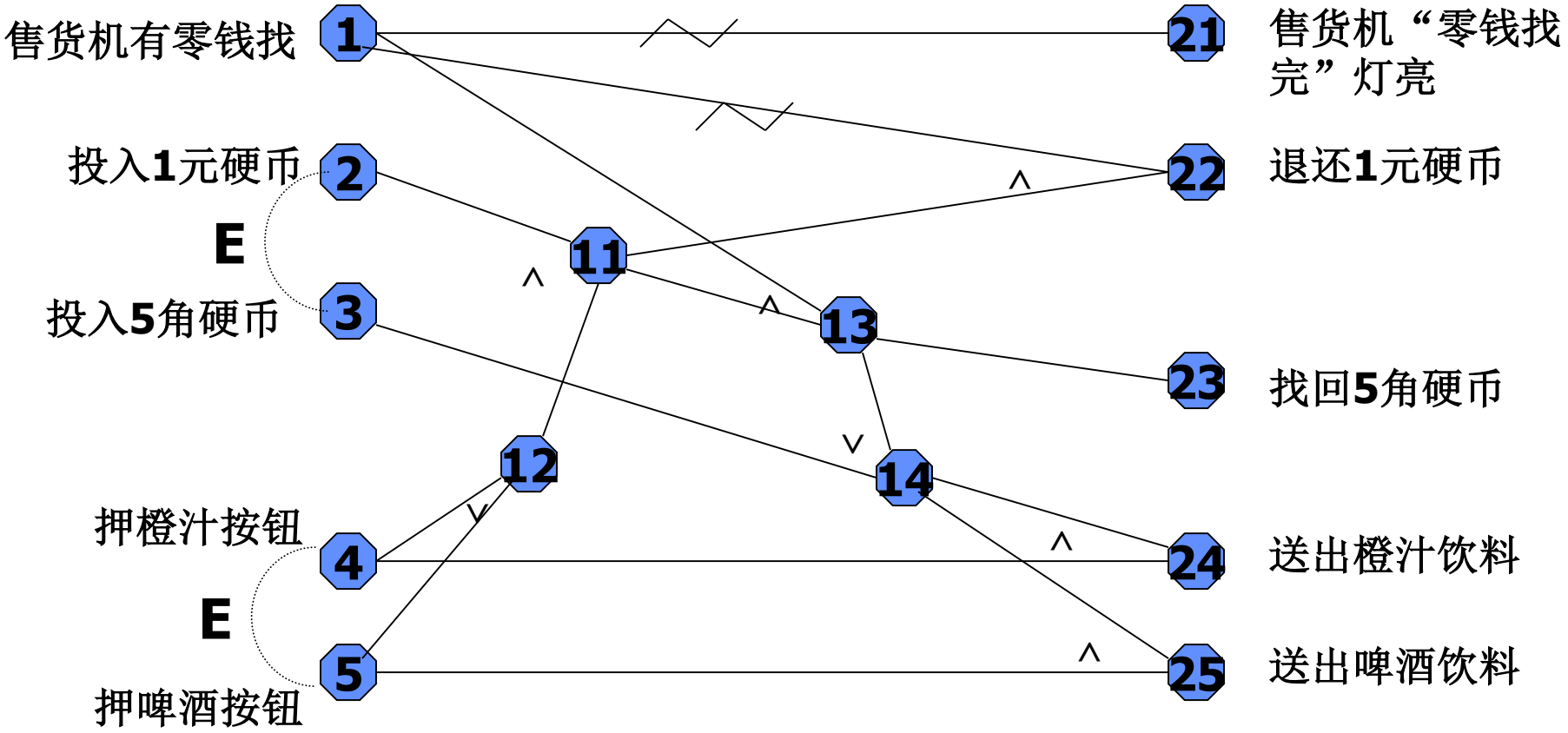
因果图测试步骤

- 测试步骤
  - 分析软件规格说明描述中，哪些是原因（即输入条件或输入条件的等价类），哪些是结果（输出条件），并给出每个原因和结果赋予一个标识符。
  - 分析软件规格说明描述中的语义，找出原因与结果之间，原因与原因之间对应的关系，画出因果图。
  - 由于语法或环境限制，有些原因与原因，原因与结果之间的组合情况不可能出现，为表明这些特殊情况，在因果图上用一些记号标明约束或限制条件。
  - 把因果图转换为判定表。
  - 把判定表中每一列表示的情况写成测试用例。

实例：因果图

有一个处理单价为5角的饮料的自动售货机，其规格说明为：若投入5角或1元的硬币，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。

例 因果图（续）


售货机有零钱找
投入1元硬币
E
投入5角硬币
押橙汁按钮
E
押啤酒按钮
1
2
3
4
5
11
12
13
14
21
22
23
24
25
售货机“零钱找完”灯亮
退还1元硬币
找回5角硬币
送出橙汁饮料
送出啤酒饮料

生成判定表

		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
	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	1	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
	11																																0
	12																																
	13																																
	14																																
	21																																
	22																																
	23																																
	24																																
	25																																

白盒测试和黑盒测试的比较

	白盒测试	黑盒测试
程序结构	已知程序结构	未知程序结构
规模	小规模测试	大规模测试
依据	详细设计说明	需求说明、概要设计说明
面向	程序结构	输入输出接口/功能要求
适用	单元测试	组装、系统测试
测试人员	开发人员	专门测试人员/外部人员
优点	能够对程序内部的特定部位进行覆盖	能站在用户立场上进行测试
缺点	无法检验程序外部特性 不能检测对要求的遗漏	不能测试程序内部特定部位 如果文档有误，则无法发现

白盒测试和黑盒测试的选择

- 白盒测试
  - 已知产品的内部工作过程
  - 证明每种内部操作是否符合设计规格要求
  - 证明所有内部成分是否都经过检查
  - 适用于单元测试
- 黑盒测试
  - 已知产品的功能设计规格说明/用户需求
  - 证明每个实现了的功能是否符合要求
  - 适用于集成/系统测试/验收测试

软件测试技术

- 从软件质量说起
- 软件测试基础
- 测试用例设计
- 软件测试策略

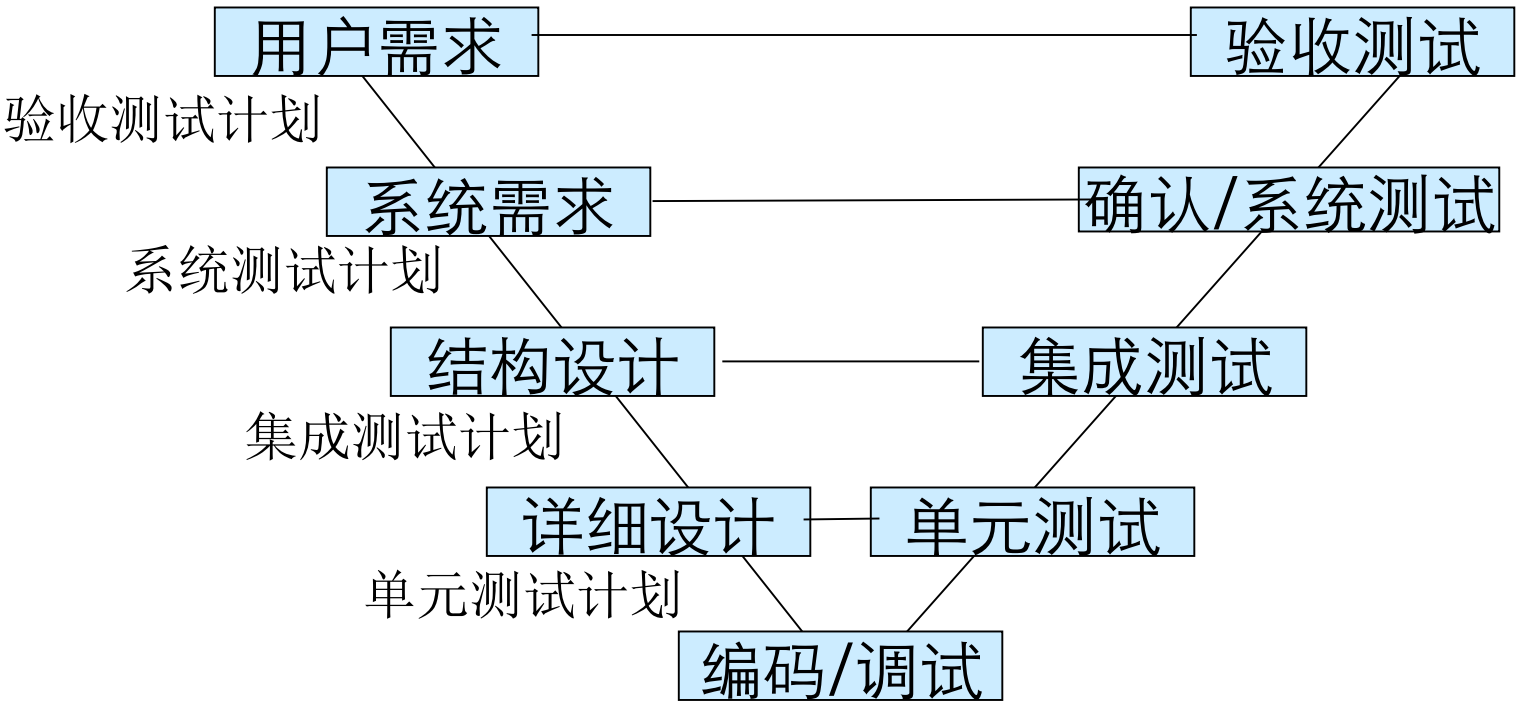
软件测试策略

- 软件测试策略

  考虑在软件开发生命周期中，软件测试的步骤、内容、方法和要求，考虑如何更有效地跟踪和控制软件产品的质量，满足用户需求。

- 典型的测试策略
  - 单元测试
  - 集成测试
  - 确认/系统测试
  - 验收测试
  - 回归测试

V模型


用户需求
验收测试
验收测试计划
系统需求
确认/系统测试
系统测试计划
结构设计
集成测试
集成测试计划
详细设计
单元测试
单元测试计划
编码/调试

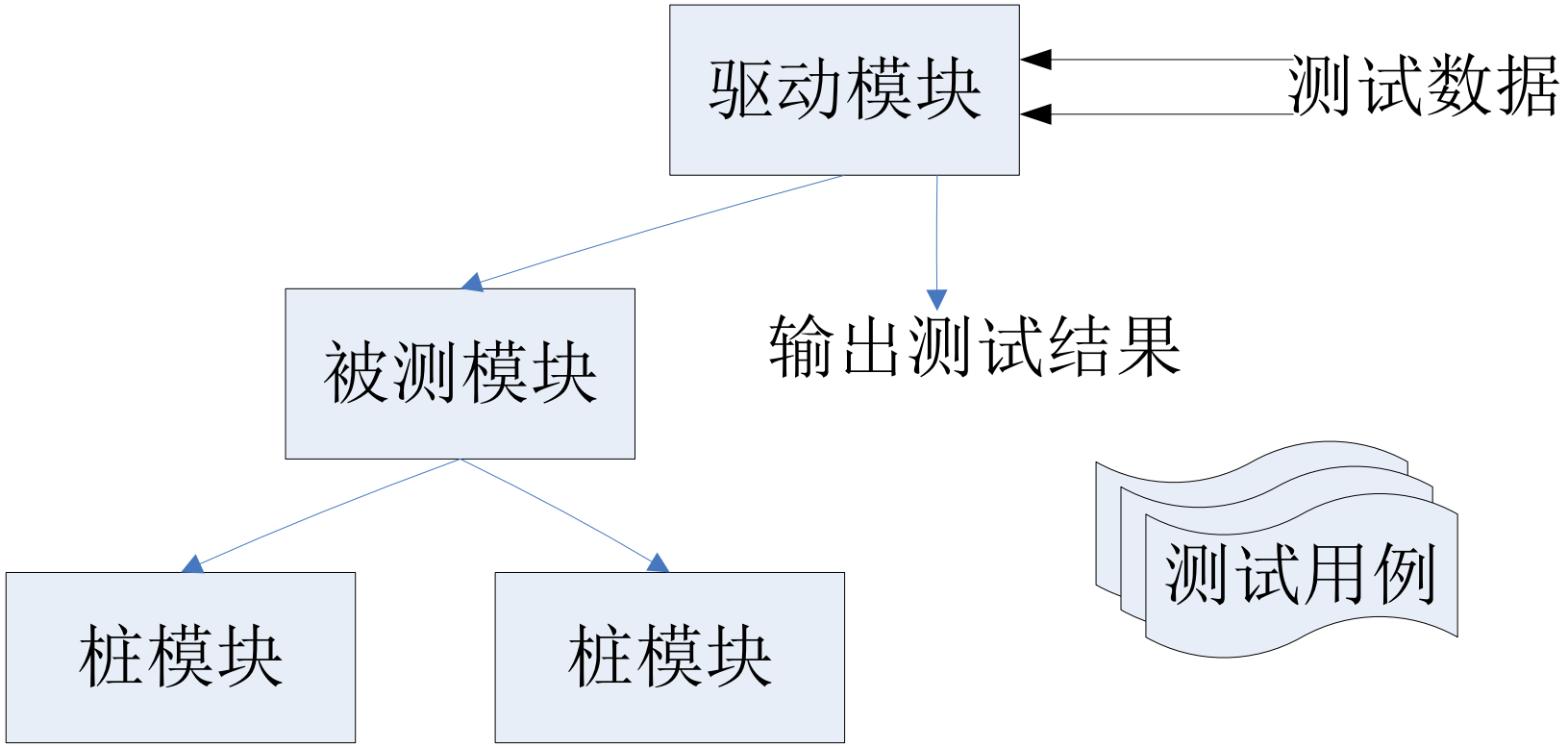
单元测试(Unit Testing)

- 单元
  - 最小软件可测试片段
  - 可编译、装配、连接、加载
- 单元测试是针对程序单元进行正确性检验的测试
  - 目的：发现各单元的功能是否满足详细设计的要求
  - 测试重点：模块内部逻辑
  - 测试技术：白盒测试

测试内容

- 模块接口测试
  - 保证在测试时进出程序单元的数据流是正确的
- 局部数据结构测试
  - 保证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
- 路径测试
  - 测试所有独立路径，保证在一个模块中的所有语句都能至少执行一次
- 边界测试
  - 保证模块在极限或严格的情况下仍能够正确执行
- 错误处理测试
  - 对不正确的输入或异常情况的处理

单元测试环境



集成测试(Integration Testing)

- 集成测试
  - 将软件单元逐步组装成系统时所进行的测试
- 目的
  - 保证低层的系统组件在组装成高层系统组件之前可正确操作，最终获得正确的软件系统
- 测试重点
  - 功能接口之间的有效性和接口的兼容性

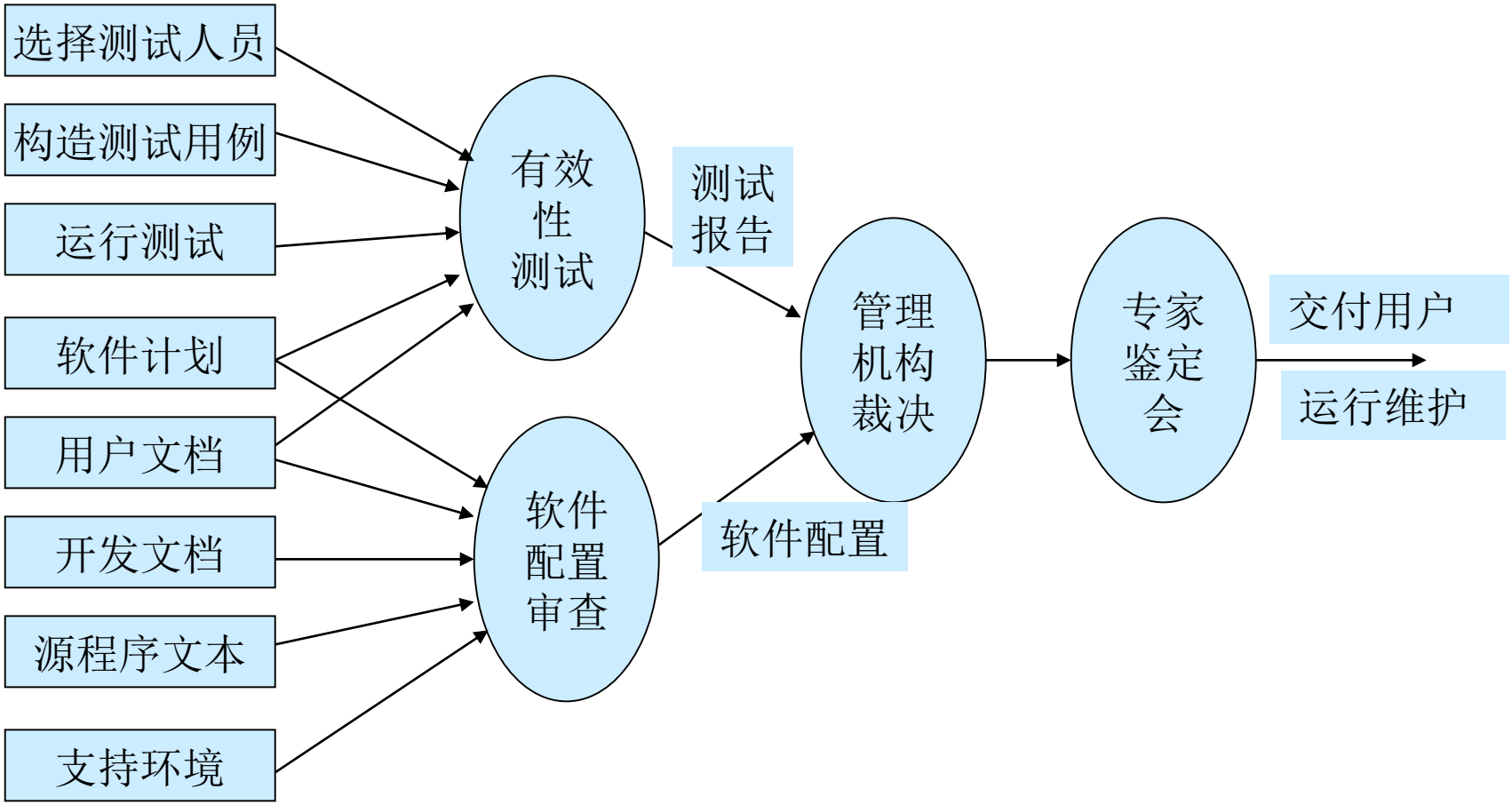
与集成有关的问题

- 配置/版本控制
- I/O接口（协议不匹配、接口参数是否正确匹配）
- 数据集成问题, 不一致的数据的显示/使用
- 错误的调用次序/错误的参数
- 遗漏/重叠的功能
- 资源问题（内存等）
- 一个模块功能是否会对另一个模块功能产生不利影响
- 各个子功能组合起来，是否达到预期要求
- 全局数据结构是否有问题
- 单个模块的误差积累起来，是否会放大，从而达到不能接受的程度。

确认测试

- 确认测试
  - **Validation testing、Qualification testing**
  - 针对全部集成好的软件系统，验证软件的功能和性能及其他特性是否与指定的要求一致
- 依据
  - 需求规格说明书或基于**SRS**的测试计划
- 确认测试中所测试的软件特性常常只有在整个系统运行时才能表现出来

确认测试步骤


选择测试人员
构造测试用例
运行测试
软件计划
用户文档
开发文档
源程序文本
支持环境
有效性测试
软件配置审查
测试报告
软件配置
管理机构裁决
专家鉴定会
交付用户
运行维护

有效性测试(黑盒测试)

- 主要目标
  - 保证所有的功能需求都得到了满足
  - 所有性能需求都达到了
  - 文档是正确且便于使用的
  - 其他需求都得到了满足
    - ✓ 可靠性、强度、
    - ✓ 可移植性、兼容性、
    - ✓ 错误恢复、可维护性

配置审查

- 配置审查
  - 功能配置审查：保证功能满足了需求，进行了充分的测试
  - 物理配置审查：保证软件产品版本一致，可以交付
    - ✓软件配置的所有成本都齐全
    - ✓质量符合要求、文档之间一致
    - ✓具有维护阶段所必须的细节
    - ✓用户文档描述的操作步骤准确、完整

Alpha测试

- **Alpha测试**
  - 由一类特定用户在开发环境下进行的测试，可以是开发机构内部的用户在模拟实际操作环境下进行的测试
  - 在系统测试证实产品达到了一定的稳定和可靠程度之后开始
  - 特点
    - ✓ 在开发者现场、由客户进行
    - ✓ 开发者：记录错误和使用问题
    - ✓ 受控的环境

Beta测试

Beta测试

- 由软件的多类用户在一类或多类用户的实际使用环境下进行的测试
- 一般在Alpha测试完成之后进行，它们主要用于面向多个用户的软件产品
- 特点
  - ✓在一类/多类客户的现场
  - ✓由最终用户进行、开发者不在现场
  - ✓客户记录问题并向开发者报告

系统测试(System Testing)

- 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，对计算机系统进行的一些列的组装和确认测试
- 目的
  - 确认整个计算机系统满足用户需求
- 依据
  - 系统需求规格说明

验收测试(Acceptance Testing)

- 验收测试
  - 以用户为主的测试，验证软件系统是否满足需求规格说明的要求
- 目的
  - 确定软件产品的质量，决定软件产品是否可以交付给用户并投入使用。
- 依据
  - 用户需求（用户需求规格说明）
- 关注
  - 需求的实现、关注隐含的需求
  - 是否“合适使用”

验收测试

- 验收测试内容
  - 功能测试
  - 性能测试
  - 可靠性测试
  - 可移植性、兼容性、可维护性、错误恢复等功能
  - 文档等
- 测试结果
  - 测试通过
  - 列出问题清单，规定在某个日期后复查
  - 测试不通过

回归测试(Regression Testing)

- 回归测试
  - 在修改之后进行，保证没有引入新的错误
  - 保证变更不会引入不期望的特性或额外的错误
  - 重新运行测试用例
    - ✓在修复/改变/增强之后
    - ✓对应用程序的每个构造重新验证所有的功能
    - ✓在修复/改变中有没有新的问题
    - ✓更完整的测试
    - ✓对构造的质量和稳定性建立信心

回归测试

- 各阶段的回归测试
  - 在单元测试期间
    - ✓ 在变更之后重新运行所有测试用例
  - 在集成测试期间
    - ✓ 重新运行所有变更单元的单元测试，以及所有集成测试
  - 在系统测试期间
    - ✓ 重新运行所有变更程序的单元测试，所有集成测试和系统测试
  - 在验收测试期间
    - ✓ 重新运行变更程序的单元测试，所有集成、系统和验收测试

测试类型

- 功能测试
  - 运行软件的所有功能，以验证这个软件系统有无严重错误
- 结构测试
  - 检查程序内部结构，包括语句、分支、路径
- 可靠性测试
  - 测试软件的可靠性指标，如平均失效间隔时间(MTBF)、平均修复时间（MTTR）
- 强度测试
  - 检查在系统运行环境不正常到发生故障的情况下，系统可以运行到何种程度的测试
- 性能测试
  - 检查系统是否满足需求说明书中规定的性能
  - 响应时间、吞吐量等

测试类型(续)

- 恢复测试
- 启动/停止测试
- 配置测试
- 安全性测试
- 压力测试
- 可使用性测试
- 可支持性测试
- 安装测试
- 互连测试
- 兼容性测试
- 文档审查

测试策略和测试类型

	开发阶段的测试					产品阶段的测试				
	设计	单元测试	模块测试	组装测试	子系统测试	有效性测试	Alpha测试	Beta测试	验收测试	系统测试
设计评审	M			S						
代码审查		M		H						S
功能测试		H	M	M	M	M	M	M	M	M
结构测试		H	M		S					
回归测试			S	H	M	M				M
可靠性测试					H	M	M	M	M	M
强度测试					H	M				H
性能测试			S		H	M	M	M	M	H
恢复测试						M				

说明：**M:**必要的　　**H:**积极推荐的　　**S:**建议使用

测试策略和测试类型(续）

	开发阶段的测试					产品阶段的测试				
	设计	单元测试	模块测试	组装测试	部件测试	有效性测试	Alpha测试	Beta测试	验收测试	系统测试
启动/停止测试						M				
配置测试					H	M				M
安全性测试						H				
可使用性测试					S	H	M	M		
可支撑性测试							H	M		
安装测试						M	M	M		
互连测试			S			M				M
兼容性测试					M	M				
压力测试					H	M				H
文档测试						M	S	H	M	