



哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY


软件工程

软件工程
第五章 软件测试
5-2 软件测试基础

徐汉川
xhc@hit.edu.cn

2017年10月23日

主要内容

- 
- 1.软件测试基础
 - 2.测试过程
 - 3.测试方法分类
 - 4.测试用例



1. 软件测试基础

1. 软件测试的概念

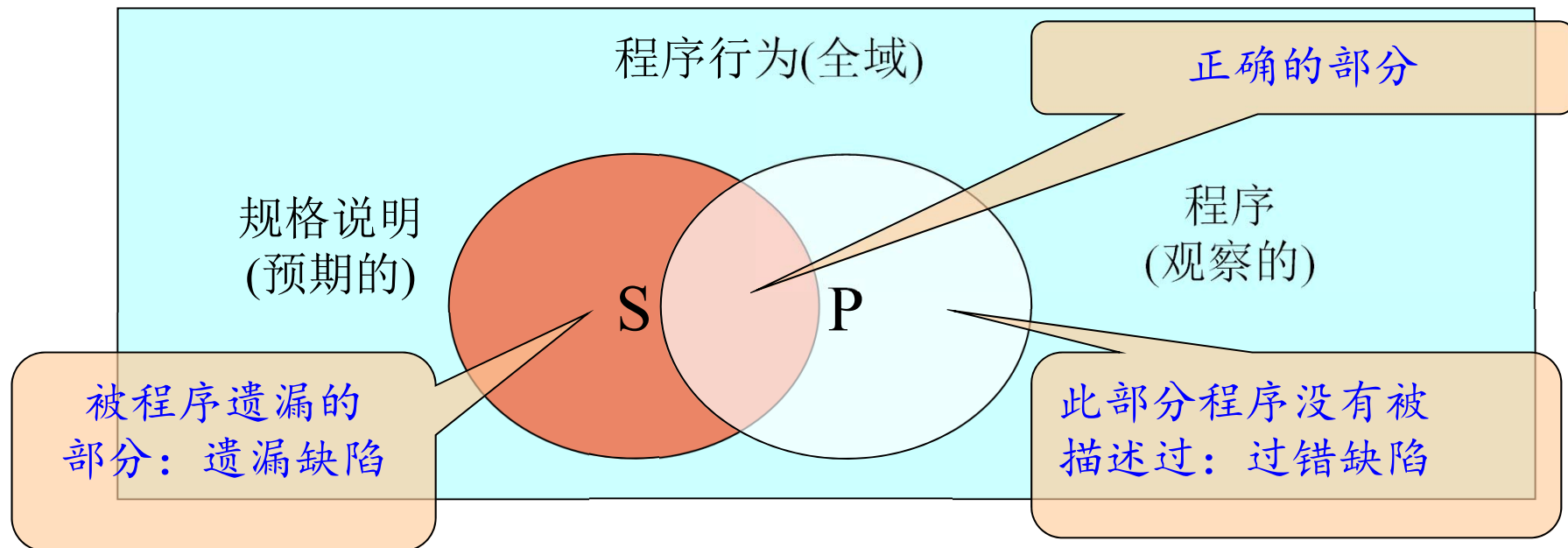
- 传统：测试是一种旨在评估一个程序或系统的属性或能力，确定它是否符合其所需结果的活动。
- **Myers**：测试是为了发现错误而执行一个程序或系统的过程。
明确提出了“在程序中寻找错误”是测试的目的。
- **IEEE**：测试是使用人工和自动手段来运行或检测某个系统的过程，其目的在于检验系统是否满足规定的需求或弄清预期结果与实际结果之间的差别。
该定义明确提出了软件测试以“检验是否满足需求”为目标。

1. 软件测试的概念

- 软件测试是更为广泛主题**验证与确认**(Verification and Validation, V&V)的一部分
 - 验证：我们在正确地构造产品吗？ 强调采用正确的手段和过程
 - 确认：我们在构造正确的产品吗？ 强调结果正确

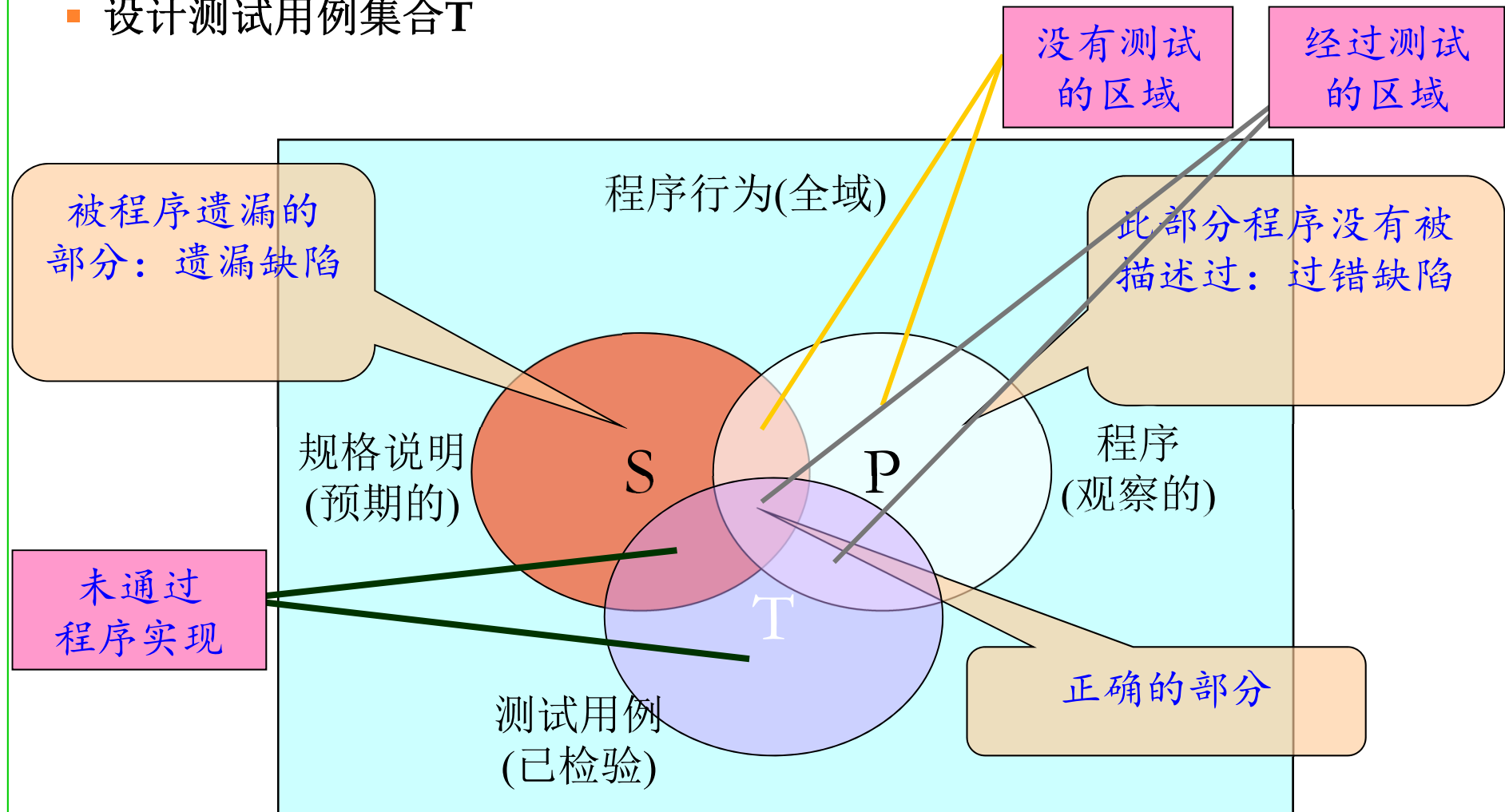
用Venn Diagram来理解测试

- 考虑一个程序行为全域，给定一段程序及其规格说明
 - 集合S是所描述的行为；
 - 集合P是用程序实现的行为；



用Venn Diagram来理解测试

- 设计测试用例集合T



软件测试的目标

- 在程序交付测试之前，大多数程序员可以找到和纠正超过**99%**的错误；
- 在交付测试的程序中，每**100**条可执行语句的平均错误数量是**1-3**个；
- 软件测试的目的就是找出剩下的**1%**的错误。

- **Glen Myers**关于软件测试目的提出以下观点：
 - 测试是为了**发现错误**而执行程序的过程
 - 测试是为了证明“程序有错”，而**无法证明**“程序正确”
 - 一个好的测试用例在于**能够发现**至今未发现的错误
 - 一个成功的测试是**发现了**至今未发现的错误的测试

2. 软件测试的原则

- 应当把“**尽早的和不断的测试**”作为软件开发者的座右铭
- **程序员应避免检查自己的程序**
- 测试**从小规模开始，逐渐扩大到大规模**
- 设计**测试用例**时，**应包括合理的输入和不合理的输入，以及各种边界条件**，特殊情况下要制造极端状态和意外状态
- 充分**注意测试中的聚集现象**：测试中发现的**80%**的错误，可能由程序的**20%**功能所造成的
- 对测试错误结果一定**要有一个确认过程**
- **制定严格的测试计划**，排除测试的随意性
- **注意回归测试的关联性**，往往修改一个错误会引起更多错误
- **妥善保存一切测试过程文档**，测试重现往往要靠测试文档

3. 软件测试文档

- **测试计划(Test Plan)**

- 测试计划是测试工作的指导性文档，规定测试活动的范围、方法、资源和进度；明确正在测试的项目、要测试的特性、要执行的测试任务、每个任务的负责人，以及与计划相关的风险。
- 主要内容：测试目标、测试方法、测试范围、测试资源、测试环境和工具、测试体系结构、测试进度表

- **测试规范(Test Specification)**

- **测试用例(Test Case)**

- **缺陷报告(Bug Report)**

3. 软件测试文档

- 测试计划(Test Plan)
- 测试规范(Test Specification)
 - 测试规范是从整体上规定测试案例的运行环境、测试方法、生成步骤、执行步骤以及调试和验证的步骤。
 - 主要内容：系统运行环境、总体测试方法、测试用例的生成步骤、测试用例的执行步骤、调试和验证
- 测试用例(Test Case)
- 缺陷报告(Bug Report)

3. 软件测试文档

- 测试计划(Test Plan)
- 测试规范(Test Specification)
- 测试用例(Test Case)
 - 测试用例是数据输入和期望结果组成的对，其中“输入”是对被测软件接收外界数据的描述，“期望结果”是对于相应输入软件应该出现的输出结果的描述，测试用例还应明确指出使用具体测试案例产生的测试程序的任何限制。
 - 测试用例可以被组织成一个测试系列，即为实现某个特定的测试目的而设计的一组测试用例。例如，一部分测试用例用来测试系统的兼容性，另一部分是用来测试系统在特定的环境中，系统的典型应用是否能够很好地运作。
- 缺陷报告(Bug Report)

3. 软件测试文档

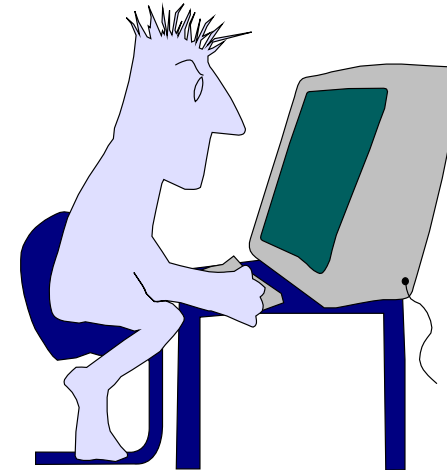
- 测试计划(Test Plan)
- 测试规范(Test Specification)
- 测试用例(Test Case)
- 缺陷报告(Bug Report)
 - 缺陷报告是编写在需要调查研究的测试过程期间发生的任何事件，简而言之，就是记录软件缺陷。
 - 主要内容：缺陷编号、题目、状态、提出、解决、所属项目、测试环境、缺陷报告步骤、期待结果、附件
 - 在报告缺陷时，一般要讲明缺陷的严重性和优先级。
 - 严重性表示缺陷的恶劣程度，反映其对产品和用户的影响。
 - 优先级表示修复缺陷的重要程度和应该何时修复。

4. 软件测试组织



开发者

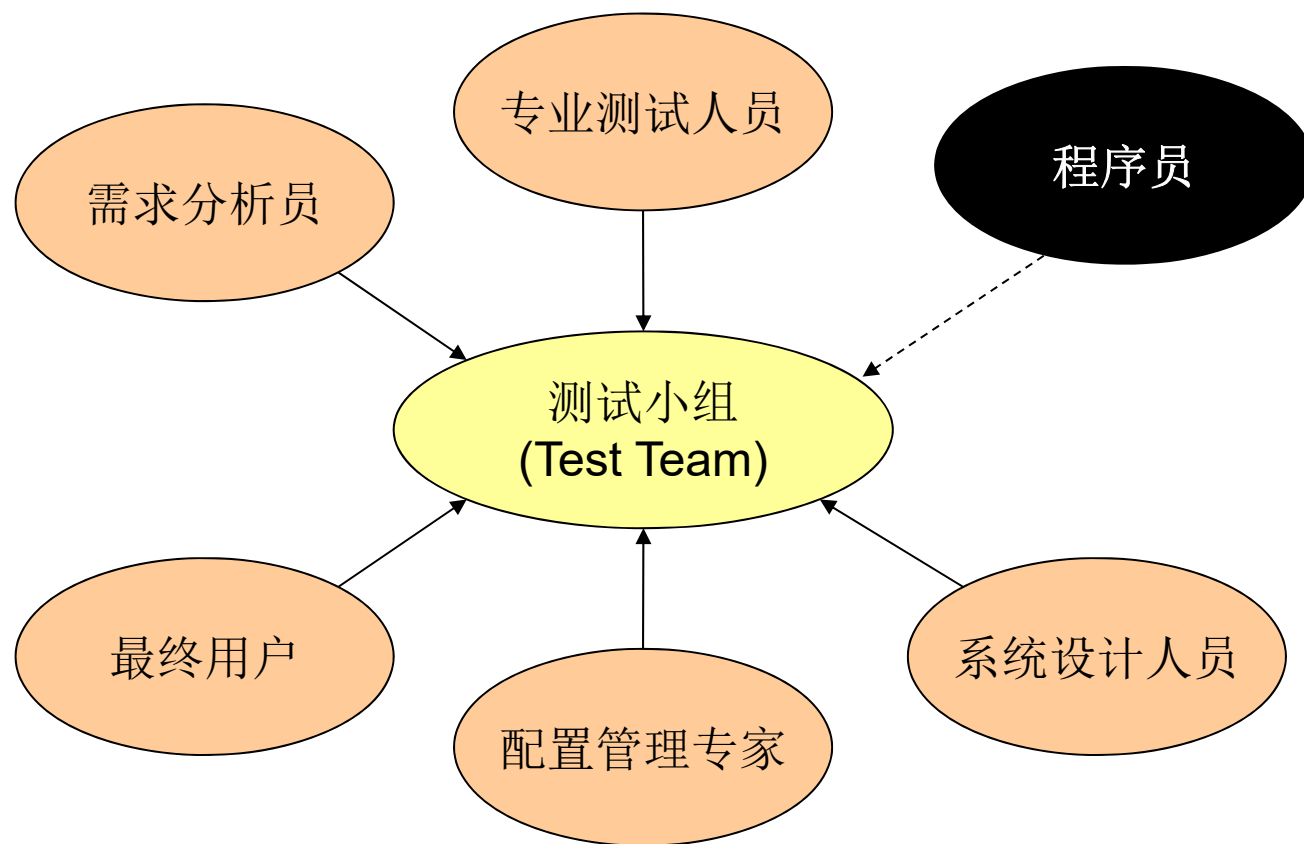
- 系统的构建者，理解系统
- 测试是“温和”的，试图证明正确性
- 发布驱动



独立测试组

- 必须学会系统
- 测试是破坏性的，试图证明错误存在
- 质量驱动

4. 软件测试组织



软件测试人员的素质要求

■ 沟通能力

- 理想的测试人员必须能与测试涉及到的所有人进行沟通，具有与技术人员(开发者)和非技术人员(客户、管理人员)的交流能力。

■ 移情能力

- 和系统开发有关的所有人员(用户、开发者、管理者)都处于一种既关心又担心的状态中。测试人员必须和每一类人打交道，因此需要对每一类人都具有足够的理解和同情，从而将测试人员与相关人员之间的冲突和对抗减少到最低程度。

■ 技术能力

- 一个测试人员必须既明白被测软件系统的概念又要会使用工程中的那些工具，最好有几年以上的编程经验，从而有助于对软件开发过程的较深入理解。

软件测试人员的素质要求

■ 自信心

- 开发人员指责测试人员出了错是常有的事，测试人员必须对自己的观点有足够的自信心。

■ 外交能力

- 当你告诉某人他出了错时，就必须使用一些外交方法，机智老练和外交手法有助于维护与开发人员之间的协作关系。

■ 幽默感

- 在遇到狡辩的情况下，一个幽默的批评将是很有帮助的。

■ 很强的记忆力

- 理想的测试人员应该有能力将以前曾经遇到过的类似的错误从记忆深处挖掘出来，这一能力在测试过程中的价值是无法衡量的。

软件测试人员的素质要求

■ 耐心

- 一些质量保证工作需要难以置信的耐心，有时需要花费惊人的时间去分离、识别一个错误。

■ 怀疑精神

- 开发人员会尽他们最大的努力将所有的错误解释过去，测试人员必须听每个人的说明，但他必须保持怀疑直到他自己看过以后。

■ 自我督促

- 干测试工作很容易变得懒散，只有那些具有自我督促能力的人才能够使自己每天正常地工作。

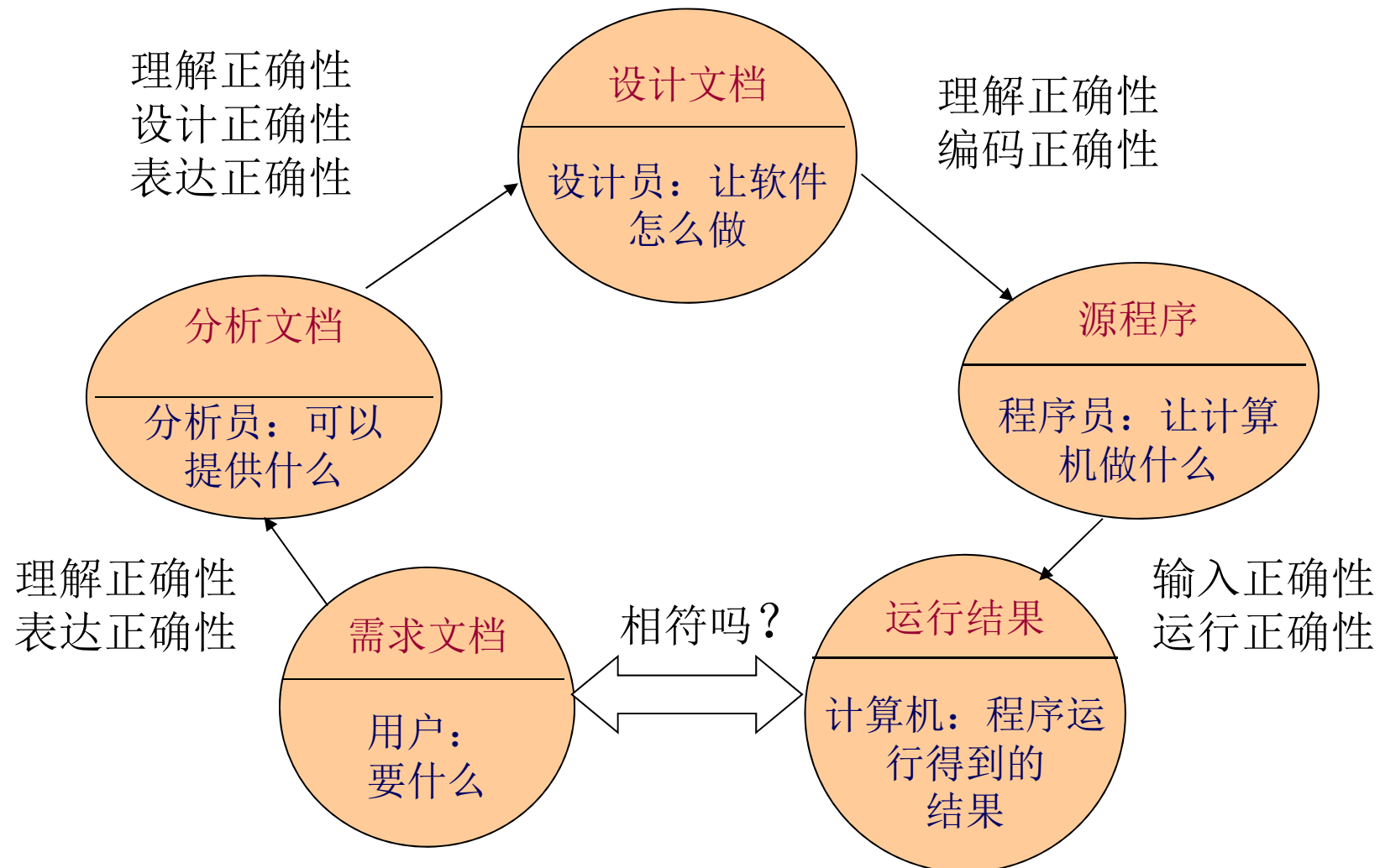
■ 洞察力

- 一个好的测试人员具有“测试是为了破坏”的观点、捕获用户观点的能力、强烈的质量追求、对细节的关注能力。

5. 软件测试的对象

- 软件测试并不等于程序测试，应贯穿于软件定义与开发的各个阶段。
- 测试对象包括：
 - 需求规格说明
 - 设计规格说明
 - 源程序

软件测试对象之间关系

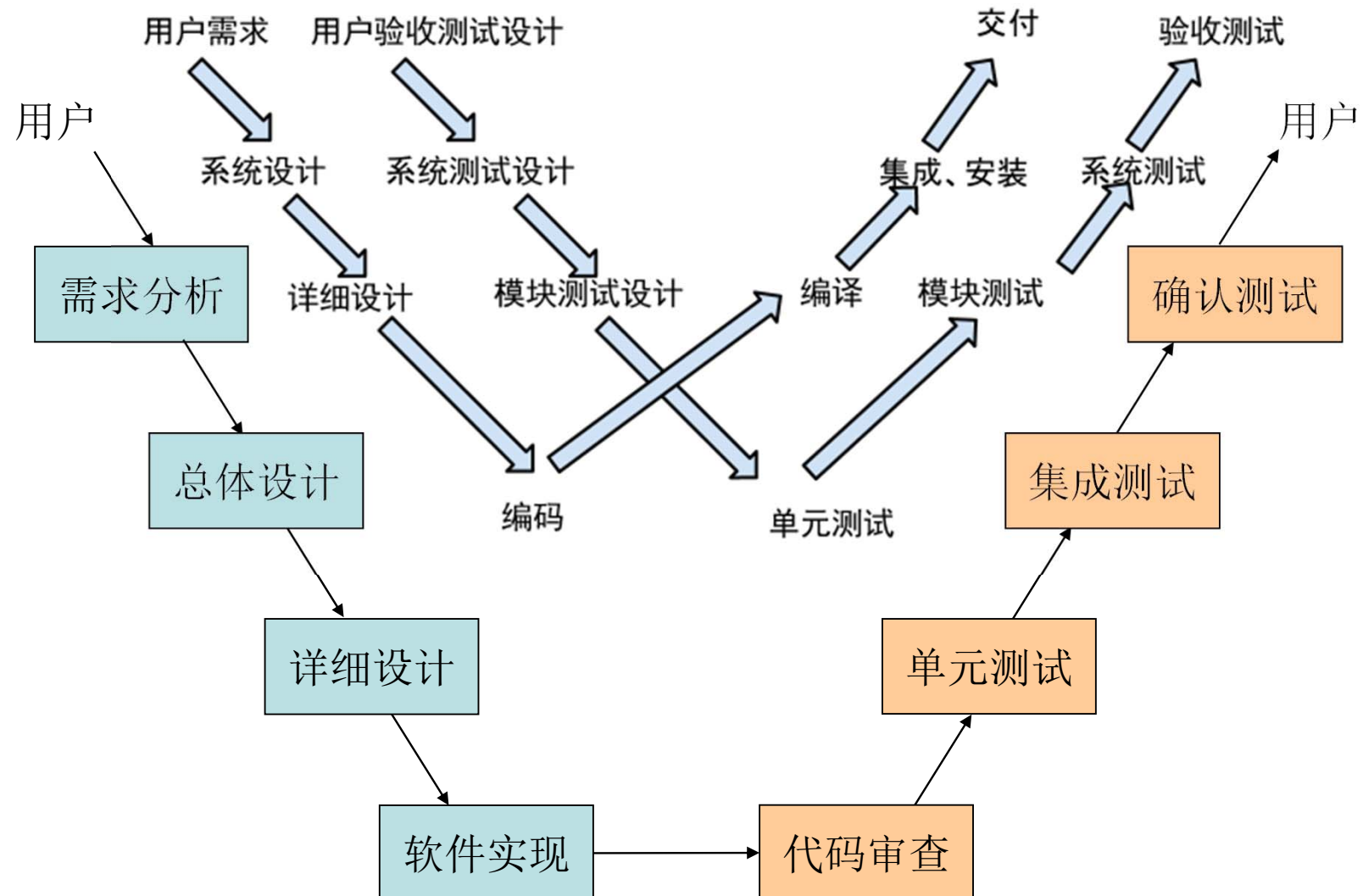




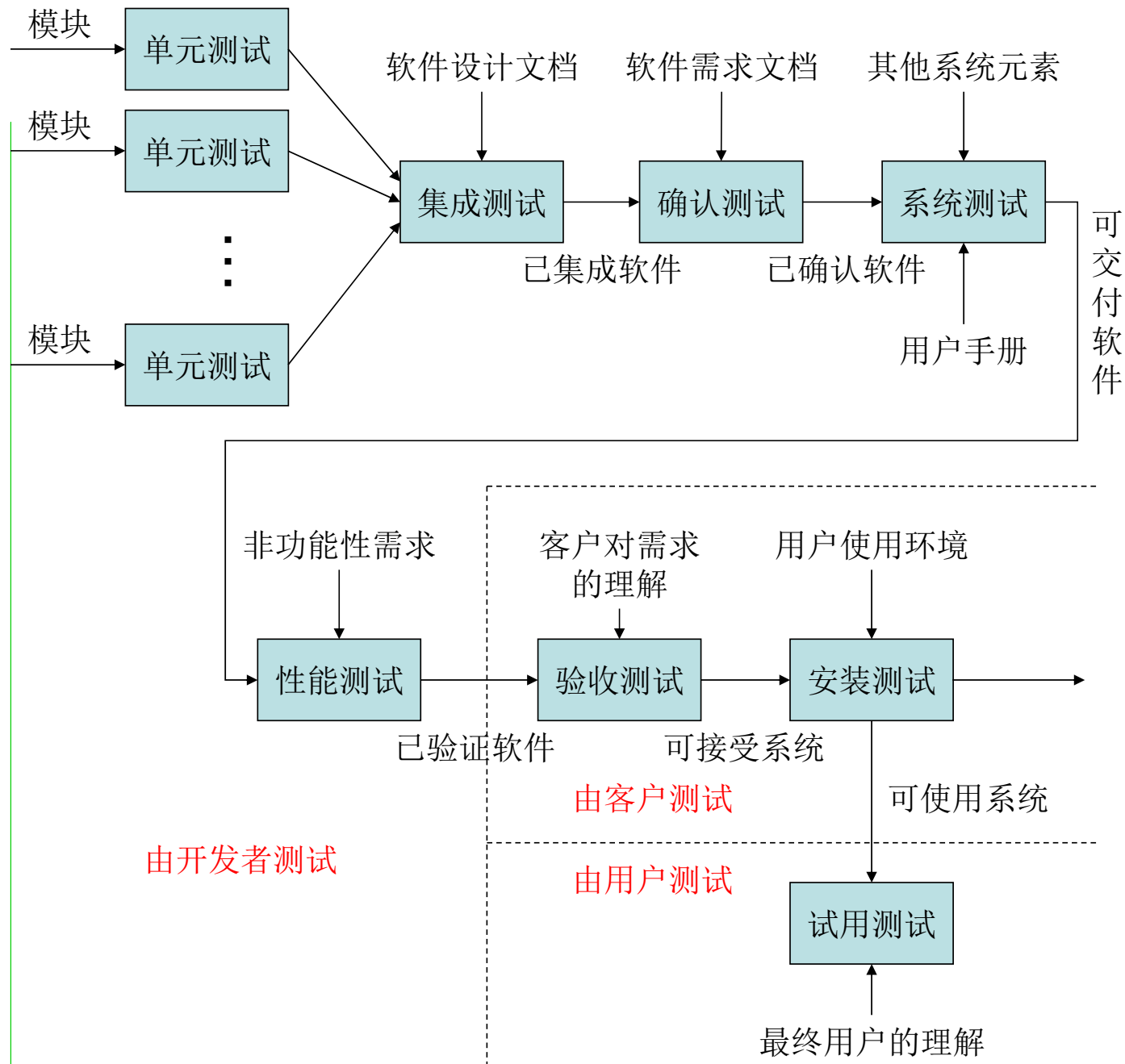
2. 软件测试过程



软件测试的V模型



软件测试活动





3. 软件测试方法的分类



软件测试方法分类

- 按实施步骤分：
 - 单元测试 Unit Testing
 - 集成测试 Integration Testing
 - 确认测试 Validation Testing
 - 系统测试 System Testing

- 按使用的测试技术分：
 - 静态测试：走查/评审
 - 动态测试：白盒/黑盒

- 按软件组装策略分：
 - 非增量测试：整体集成
 - 增量测试：自顶向下、自底向上、三明治

1. 单元测试(Unit Testing)

■ 单元测试目的

- 验证开发人员所书写的代码是否可以按照其所设想的方式执行而产出符合预期值的结果，确保产生符合需求的可靠程序单元。

■ 单元测试范围

- 单元测试是对软件基本组成单元（构件或模块）进行的测试
- 单元测试侧重于构件中的内部处理逻辑和数据结构（依据详细设计）
- 结构化程序单元是模块，面向对象程序单元是类

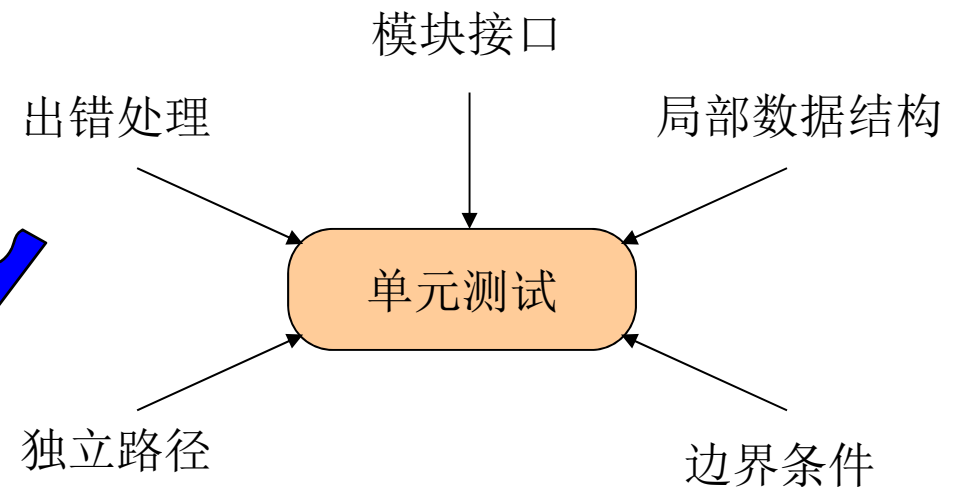
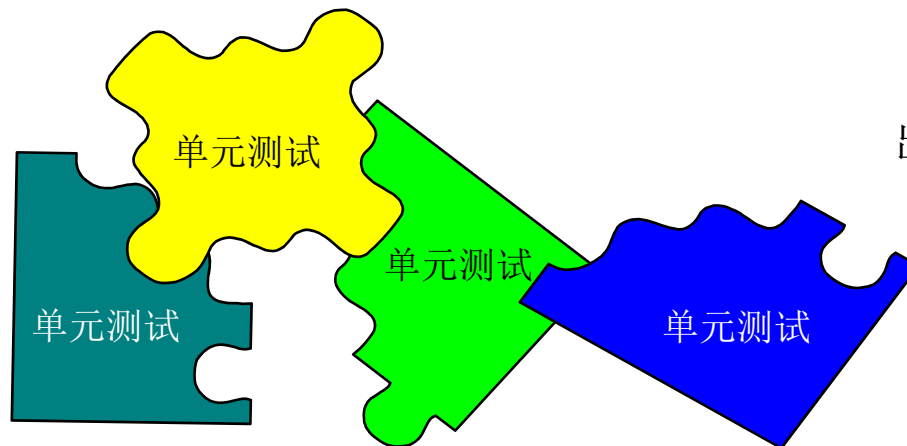
■ 单元测试规程

- 单元测试通常被认为是编码阶段的附属工作，单元测试可以在编码开始之前或源代码生成之后完成
- 单元测试一般由编写该单元代码的开发人员执行，该人员负责设计和运行一系列的测试以确保该单元符合需求。

单元测试

■ 单元测试内容

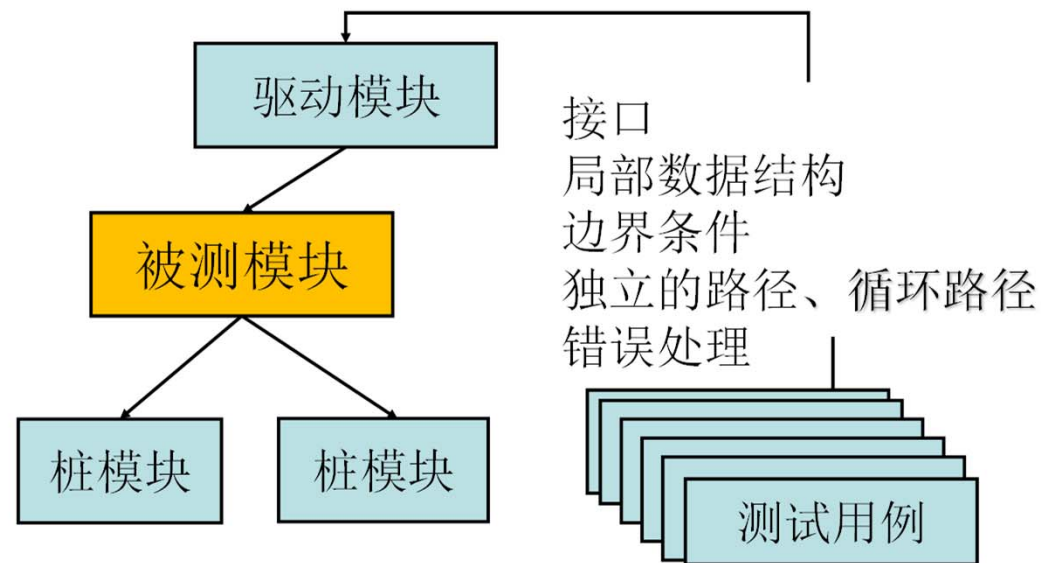
- 接口
- 局部数据结构
- 独立路径
- 边界条件
- 错误处理路径



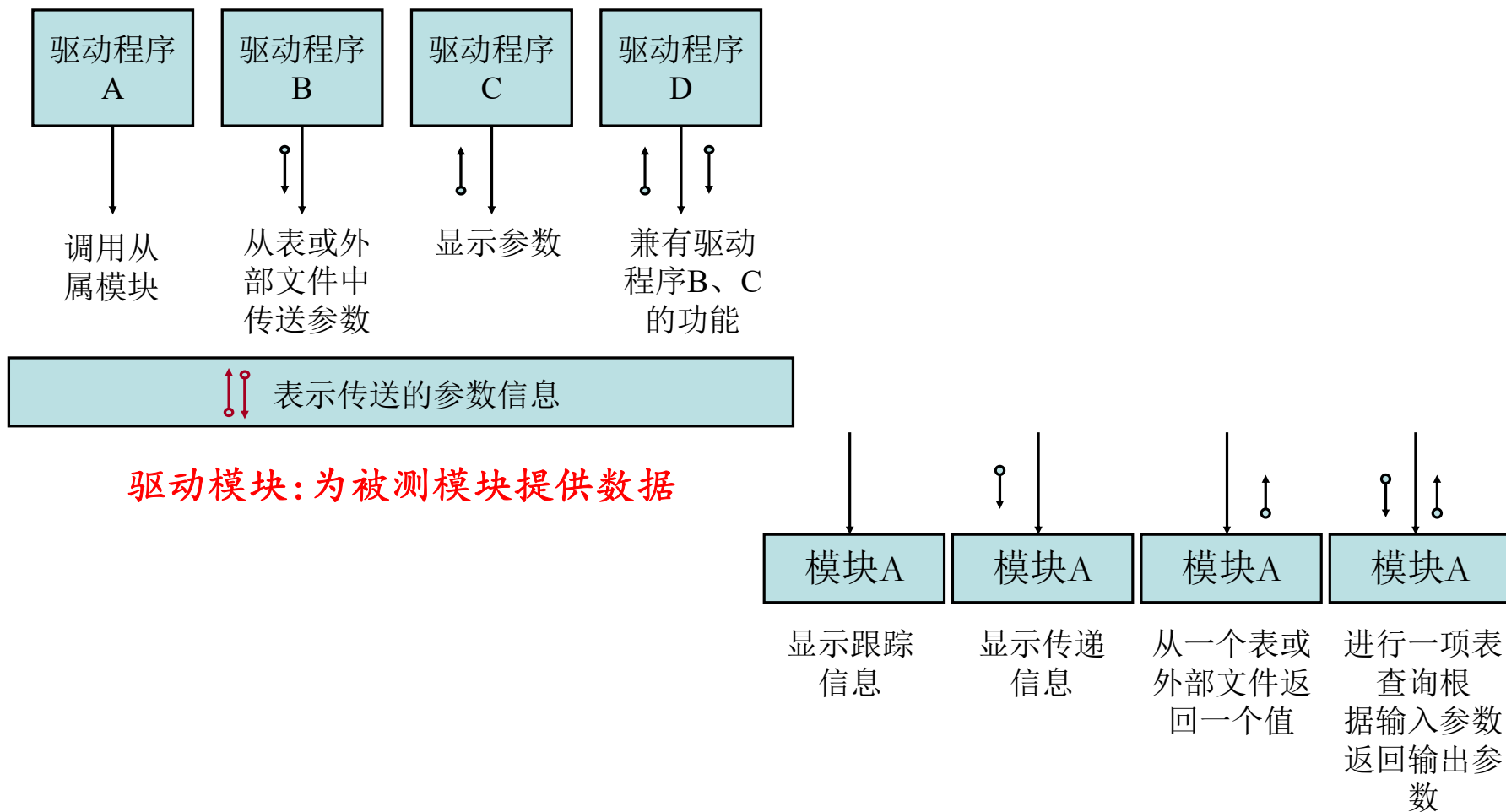
单元测试的环境

■ 单元测试环境

- **驱动模块(driver)**: 模拟被测模块的上一级模块, 接收测试数据, 把这些数据传送给所测模块, 最后再输出实际测试结果;
- **桩模块(stub)**: 模拟被测单元需调用的其他函数接口, 模拟实现子函数的某些功能。



单元测试的驱动模块/桩模块



驱动模块: 为被测模块提供数据

桩模块: 只做少量的数据操作

2. 集成测试

- 每个模块都能单独工作，但是集成在一起却不能工作？ Why？
 - 模块通过接口相互调用时会引入很多新问题...
- 集成测试(Integration Testing)
 - 在单元测试的基础上，将所有模块按照总体设计的要求组装成为子系统或系统进行的测试。
 - 集成测试是构造软件体系结构的系统化技术，同时也是进行一些旨在发现与接口相关的错误的测试。
 - 结构化集成测试针对调用关系测试，面向对象针对依赖关系测试

集成测试策略

■ 集成测试策略

- 基于层次的集成：自顶向下与自底向上
- 基于功能的集成：按照功能的优先级逐步将模块加入系统中
- 基于进度的集成：把最早可获得的代码进行集成
- 基于使用的集成：通过类的使用关系进行集成

集成测试的目标

■ 集成测试考虑的问题：

- 模块接口的数据是否会丢失
- 组合后的子功能，能否达到预期要求的父功能
- 模块的功能是否会相互产生不利的影响
- 全局数据结构是否有问题
- 模块的误差累积是否会放大
- 单个模块的错误是否会导致数据库错误

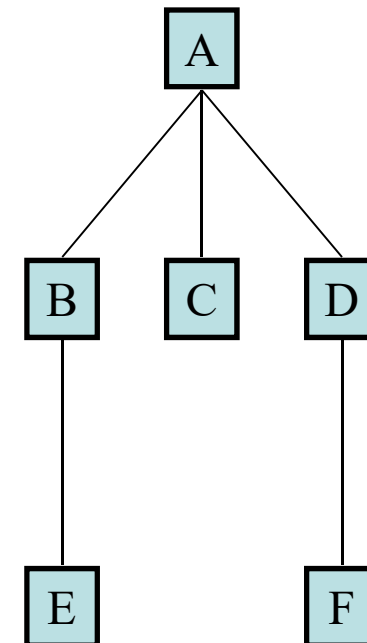
集成测试的方法：整体集成

■ 整体集成方式(非增量式集成)

- 把所有模块按设计要求一次全部组装起来，然后进行整体测试
- “一步到位”的集成方式

■ 例如：

- Test A (with stubs for B, C, D)
- Test B (with driver for A and stub for E)
- Test C (with driver for A)
- Test D (with driver for A and stub for F)
- Test E (with driver for B)
- Test F (with driver for D)
- Test (A, B, C, D, E, F)



集成测试的方法：整体集成

- 优点：

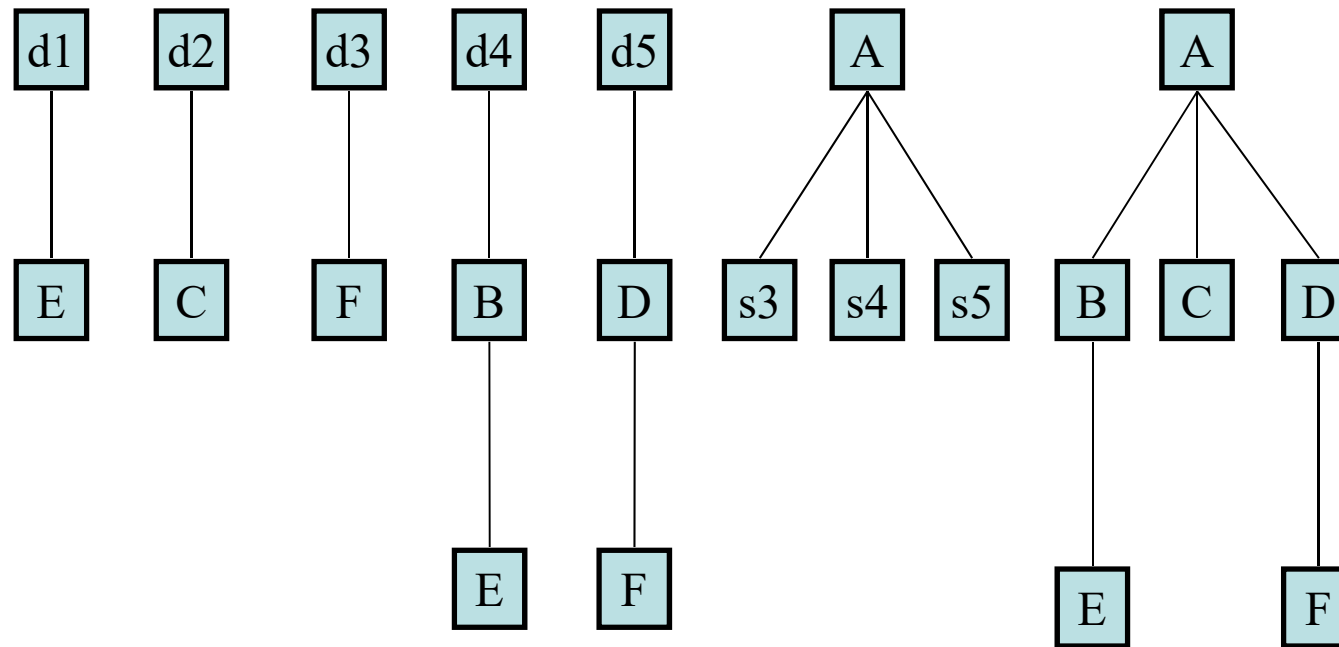
- 效率高，所需人力资源少；
- 测试用例数目少，工作量低；
- 简单，易行；

- 缺点：

- 可能发现大量的错误，难以进行错误定位和修改；
- 即使测试通过，也会遗漏很多错误；
- 测试和修改过程中，新旧错误混杂，带来调试困难；

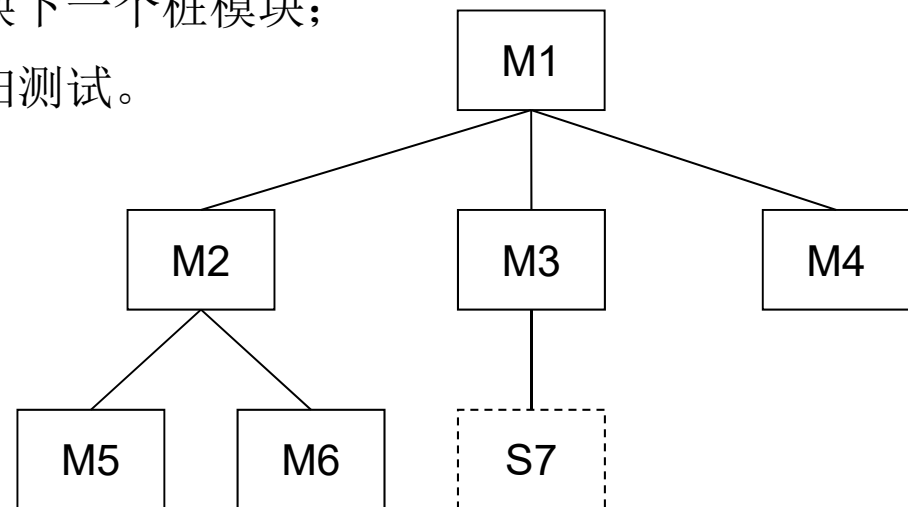
集成测试的方法：增量式集成

- 增量式集成测试方法：
 - 逐步将新模块加入并测试



(1) 自顶向下的增量集成

- **自顶向下的集成测试：**从主控模块开始，按软件的控制层次结构，以深度优先或广度优先的策略，逐步把各个模块集成在一起。
- **具体步骤：**
 - 1：以主控模块作为测试驱动模块，把对主控模块进行单元测试时所引入的所有桩模块用实际模块代替；
 - 2：依据所选的集成策略(深度优先、广度优先)，每次只替代一个桩模块；
 - 3：每集成一个模块立即测试一遍；
 - 4：只有每组测试完成后，才着手替换下一个桩模块；
 - 5：为避免引入新错误，不断进行回归测试。



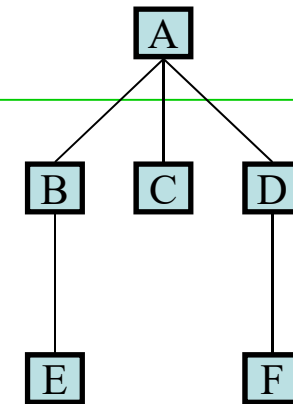
(1) 自顶向下的增量集成

■ 自顶向下集成

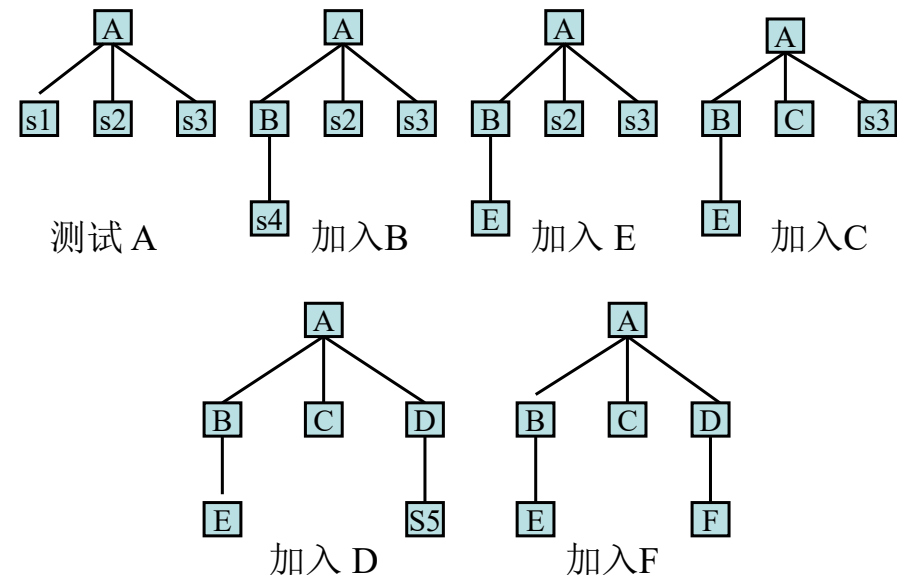
- 深度优先：A、B、E、C、D、F
- 广度优先：A、B、C、D、E、F

■ 广度优先的测试过程：

- Test A (with stubs for B,C,D)
- Test A;B (with stubs for E,C,D)
- Test A;B;C (with stubs for E,D)
- Test A;B;C;D (with stubs for E,F)
- Test A;B;C;D;E (with stubs for F)
- Test A;B;C;D;E;F

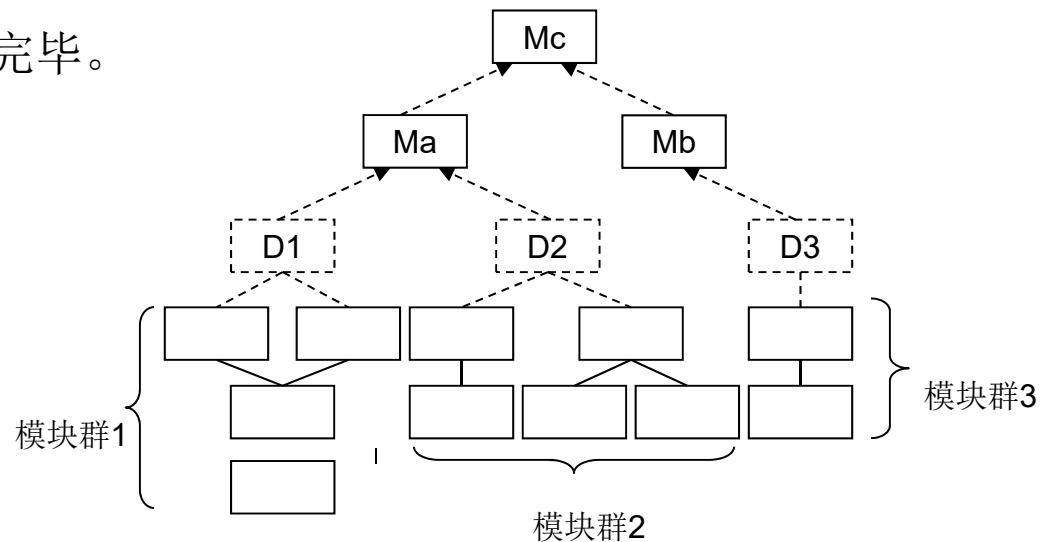


深度优先的测试过程



(2) 自底向上的增量集成

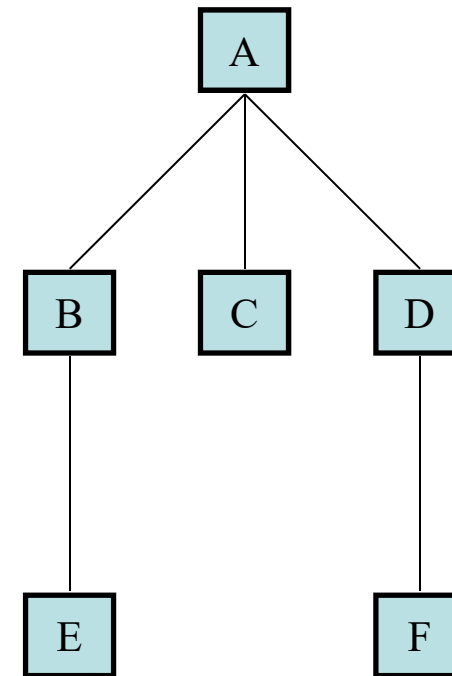
- **自底向上的集成测试：**从软件结构最底层的模块开始组装测试。
- **具体步骤：**
 - 1：把底层模块组织成实现某个子功能的模块群(cluster)；
 - 2：开发一个测试驱动模块，控制测试数据的输入和测试结果的输出；
 - 3：对每个模块群进行测试；
 - 4：删除测试使用的驱动模块，用较高层模块把模块去组织成为完成更大功能的新模块；
 - 5：循环，直到整个程序测试完毕。



(2) 自底向上的增量集成

■ 过程:

- Test E (with driver for B)
- Test C (with driver for A)
- Test F (with driver for D)
- Test B;E (with driver for A)
- Test D;F (with driver for A)
- Test (A;B;C;D;E;F)



两种集成测试的优缺点

■ 自顶向下集成：

- 优点：能尽早地对程序的主要控制和决策机制进行检验，因此可较早地发现错误；软件某个完整功能可以被实现和展示（先深集成策略）；较少需要驱动模块；
- 缺点：所需的桩模块数量巨大；在测试较高层模块时，低层处理采用桩模块替代，不能反映真实情况，重要数据不能及时回送到上层模块，因此测试并不充分；

■ 自底向上集成：

- 优点：不用桩模块，测试用例的设计亦相对简单；
- 缺点：程序最后一个模块加入时才具有整体形象，难以尽早建立信心。

(3) 三明治集成

- **三明治集成**：一种混合增量式集成策略，综合了自顶向下和自底向上两种方法的优点。
- **具体步骤**：
 - S1：确定以哪一层为界来决定使用三明治集成策略；
 - S2：对该层次及其下面的所有各层使用自底向上的集成策略；
 - S3：对该层次之上的所有各层使用自顶向下的集成策略；
 - S4：把该层次各模块同相应的下层集成；
 - S5：对系统进行整体测试。

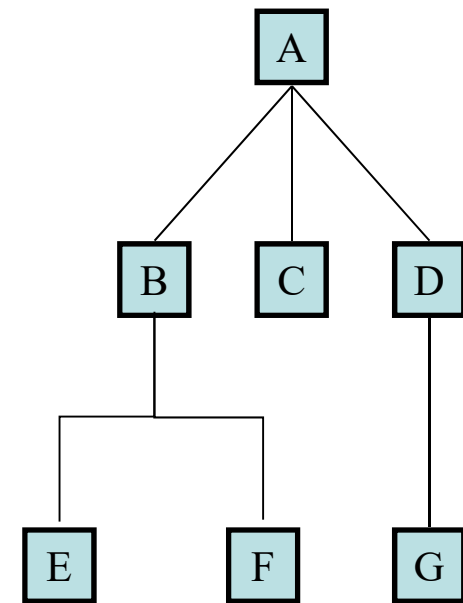
(3) 三明治集成

- 选定模块**B**所在的中间层，过程如下：

- Test A (with stubs for B,C,D)
- Test B (with driver for A and stubs for E,F)
- Test C (with driver for A)
- Test D (with driver for A and stub for G)
- Test A;B (with stubs for E,F,C,D)
- Test A;B;C (with stubs for E,F,D)
- Test A;B;C;D (with stubs for E,F,G)
- Test E (with driver for B)
- Test F (with driver for B)
- Test B;E;F (with driver for A)
- Test G (with driver for D)
- Test D;G (with driver for A)
- Test A;B;C;D;E;F;G

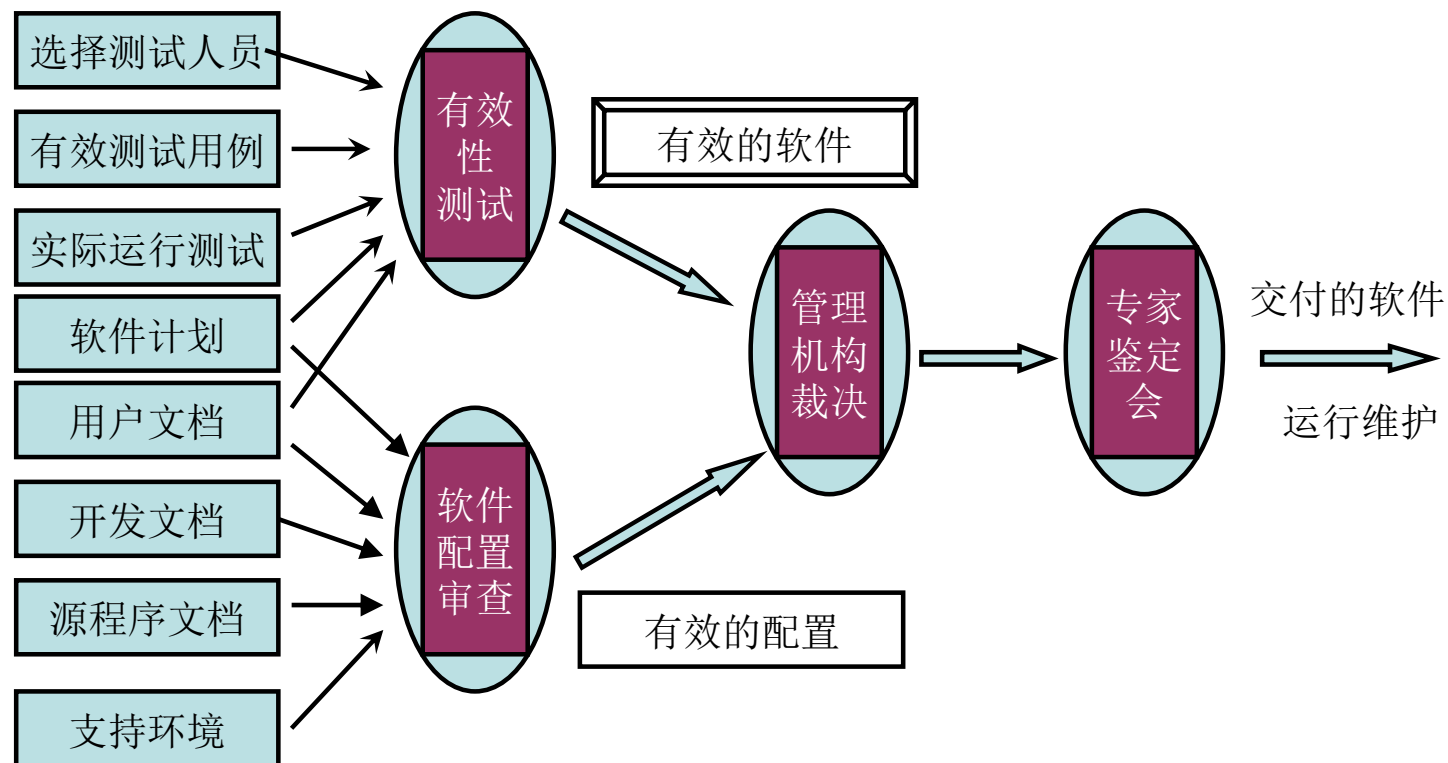
自顶向下

自底向上



3. 确认测试

- 软件确认是通过一系列表明已符合软件需求的测试而获得的
- 确认测试(Validation Testing)：检查软件能否按合同要求进行工作，即是否满足软件需求说明书中的确认标准。



4. 系统测试

■ 系统测试(System Testing)

- 系统测试是将已经集成好的软件系统作为一个元素，与计算机硬件、外设、某些支持软件、数据和人员等其他元素结合在一起，在实际运行环境下进行的一系列测试。

■ 系统测试方法

- 功能测试、协议一致性测试
- 性能测试、压力测试、容量测试、安全性测试、恢复性测试
- 备份测试、GUI 测试、健壮性测试、兼容性测试、可用性测试
- 可安装性测试、文档测试、在线帮助测试、数据转换测试

系统测试：功能测试

■ 功能测试(Functional Testing)

- 功能测试是系统测试中最基本的测试，它不管软件内部的实现逻辑，主要根据软件需求规格说明和测试需求列表，验证产品的功能实现是否符合需求规格。
- 功能测试主要发现以下错误：
 - 是否有不正确或遗漏的功能？
 - 功能实现是否满足用户需求和系统设计的隐藏需求？
 - 能否正确地接受输入？能否正确地输出结果？
- 常用的测试技术
 - 黑盒测试方法：等价类划分、边界值测试

系统测试：压力测试

■ 压力测试(Press Testing)

- 压力测试是检查系统在资源超负荷情况下的表现，特别是对系统的处理时间有什么影响。
- 压力测试的例子
 - 对于一个固定输入速率(如每分钟120 个单词)的单词处理响应时间
 - 在一个非常短的时间内引入超负荷的数据容量
 - 成千上万的用户在同一时间从网上登录到系统
 - 引入需要大量内存资源的操作
- 压力测试采用边界值和错误猜测方法，且需要工具的支持。

系统测试：安全性测试

■ 安全性测试(Security Testing)

- 安全性测试检查系统对非法侵入的防范能力。
- 安全性测试期间，测试人员假扮非法入侵者，采用各种办法试图突破防线。
- 安全性测试的例子
 - 想方设法截取或破译口令
 - 专门定做软件破坏系统的保护机制
 - 故意导致系统失败，企图趁恢复之机非法进入
 - 试图通过浏览非保密数据，推导所需信息

系统测试：恢复测试

■ 恢复测试(Recovery Testing)

- 恢复测试是检验系统从软件或者硬件失败中恢复的能力，即采用各种人工干预方式使软件出错，而不能正常工作，从而检验系统的恢复能力。
- 恢复性测试的例子
 - 当供电出现问题时的恢复
 - 恢复程序的执行
 - 对选择的文件和数据进行恢复
 - 恢复处理日志方面的能力
 - 通过切换到一个并行系统来进行恢复

系统测试：GUI测试

■ GUI测试(Graphic User Interface Testing)

- GUI测试一是检查用户界面实现与设计的符合情况，二是确认用户界面处理的正确性。
- GUI测试提倡界面与功能的设计分离，其重点关注在界面层和界面与功能接口层上。
- GUI自动化测试工具
 - WinRunner, QARun, QARobot, Visual Test
- 常用的测试技术
 - 等价类划分、边界值分析、基于状态图方法、错误猜测法

系统测试：安装测试

■ 安装测试(Installation Testing)

- 系统验收之后，需要在目标环境中进行安装，其目的是保证应用程序能够被成功地安装。
- 安装测试应考虑
 - 应用程序是否可以成功地安装在以前从未安装过的环境中？
 - 应用程序是否可以成功地安装在以前已有的环境中？
 - 配置信息定义正确吗？
 - 考虑到以前的配置信息吗？
 - 在线文档安装正确吗？
 - 安装应用程序是否会影响其他的应用程序吗？
 - 安装程序是否可以检测到资源的情况并做出适当的反应？

5. 验收测试

■ 验收测试(Acceptance Testing)

- 验收测试是以用户为主的测试，一般使用用户环境中的实际数据进行测试。
- 在测试过程中，除了考虑软件的功能和性能外，还应对软件的兼容性、可维护性、错误的恢复功能等进行确认。

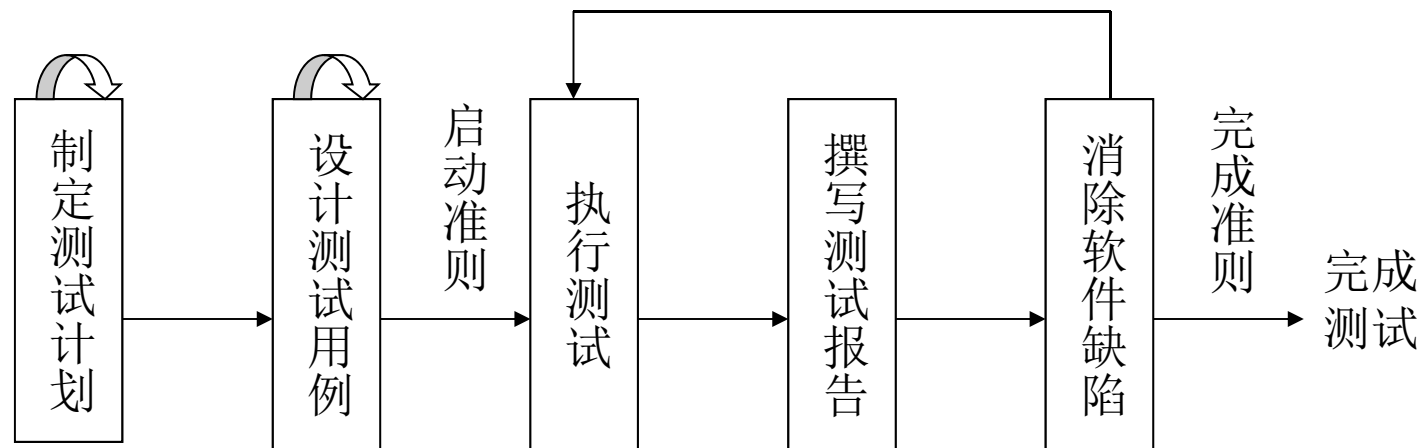
■ α 测试与 β 测试

- α 测试与 β 测试是产品在正式发布前经常进行的两种测试；
 - α 测试是由用户在开发环境下进行的测试；
 - β 测试是由软件的多个用户在实际使用环境下进行的测试。

6. 回归测试

■ 回归测试(Regression Testing)

- 回归测试是验证对系统的变更没有影响以前的功能，并且保证当前功能的变更是正确的。
- 回归测试可以发生在软件测试的任何阶段，包括单元测试、集成测试和系统测试，其令人烦恼的原因在于频繁的重复性劳动。
- 回归测试应考虑的因素
 - 范围：有选择地执行以前的测试用例；
 - 自动化：测试程序的自动执行和自动配置、测试用例的管理和自动输入、测试结果的自动采集和比较、测试结论的自动输出。



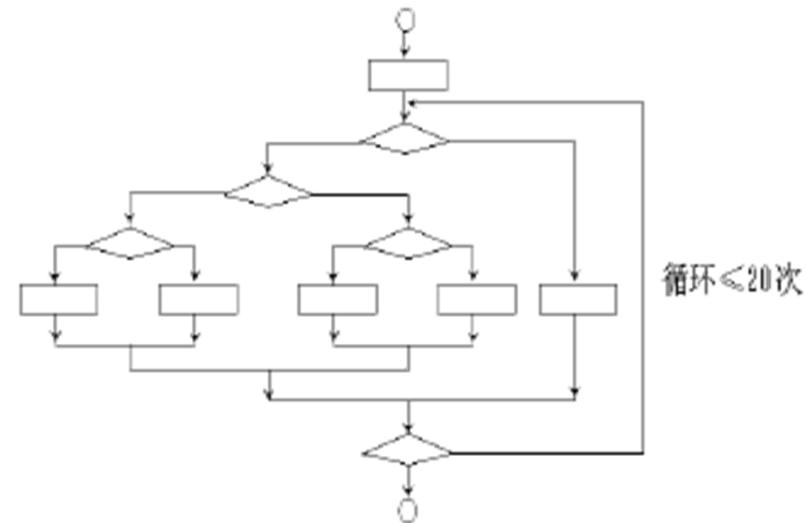
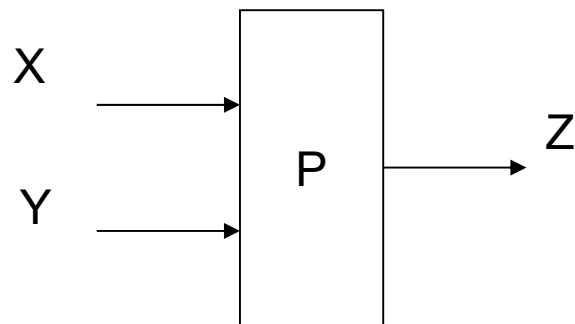
典型的软件测试技术

■ 黑盒测试：又称“功能测试”或“数据驱动测试”

- 它将测试对象看做一个黑盒子，测试人员完全不考虑程序内部的逻辑结构和内部特性，只依据程序的需求规格说明书，检查程序的功能是否符合它的功能说明。**主要用于集成测试、确认测试和系统测试。**

■ 白盒测试：又称“结构测试”或“逻辑驱动测试”

- 它把测试对象看做一个透明的盒子，它允许测试人员利用程序内部的逻辑结构及有关信息，设计或选择测试用例，对程序所有逻辑路径进行测试。**主要用于单元测试和集成测试。**





4. 测试用例



测试用例的定义与特征

■ 测试用例(testing case):

- 测试用例是为特定的目的而设计的一组测试输入、执行条件和预期的结果。
- 测试用例是执行的最小测试实体。
- 测试用例就是设计一个场景，使软件程序在这种场景下，必须能够正常运行并且达到程序所设计的执行结果。

■ 测试用例的特征:

- 最有可能抓住错误的;
- 不是重复的、多余的;
- 一组相似测试用例中最有效的;
- 既不是太简单，也不是太复杂。

测试用例的设计原则

- **测试用例的代表性：**

- 能够代表并覆盖各种合理的和不合理的、合法的和非法的、边界的和越界的以及极限的输入数据、操作和环境设置等

- **测试结果的可判定性：**

- 测试执行结果的正确性是可判定的，每一个测试用例都应有相应的期望结果。

- **测试结果的可再现性：**

- 对同样的测试用例，系统的执行结果应当是相同的。



哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

软件工程

結束

2017年10月23日