Unit Testing

单元测试的定义

定义：

单元测试是对软件基本组成单元进行的测试。

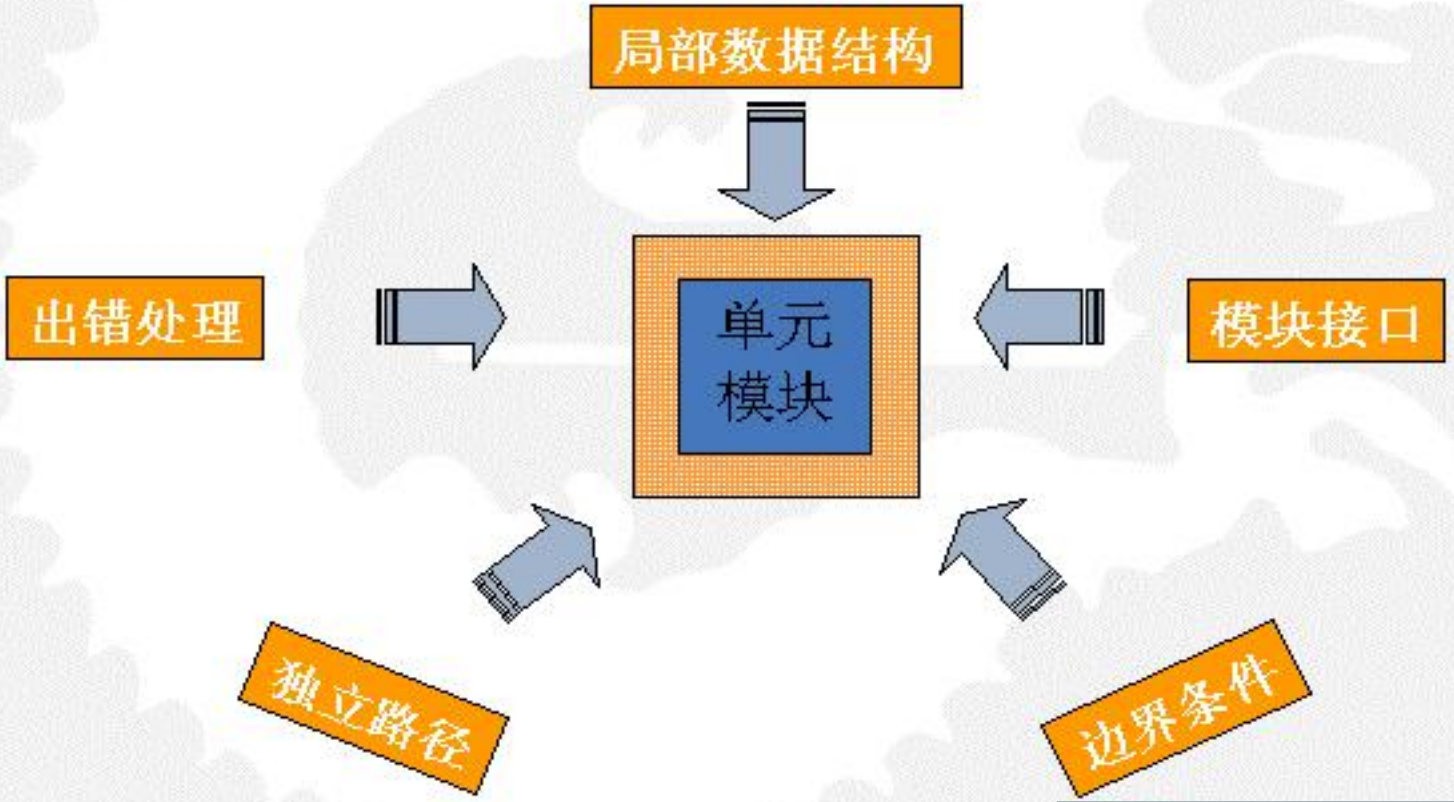
时机：

一般在代码完成后由开发人员完成,QA人员辅助.（微软的例子是例外的，根据具体的测试人员的素质与技术有关）

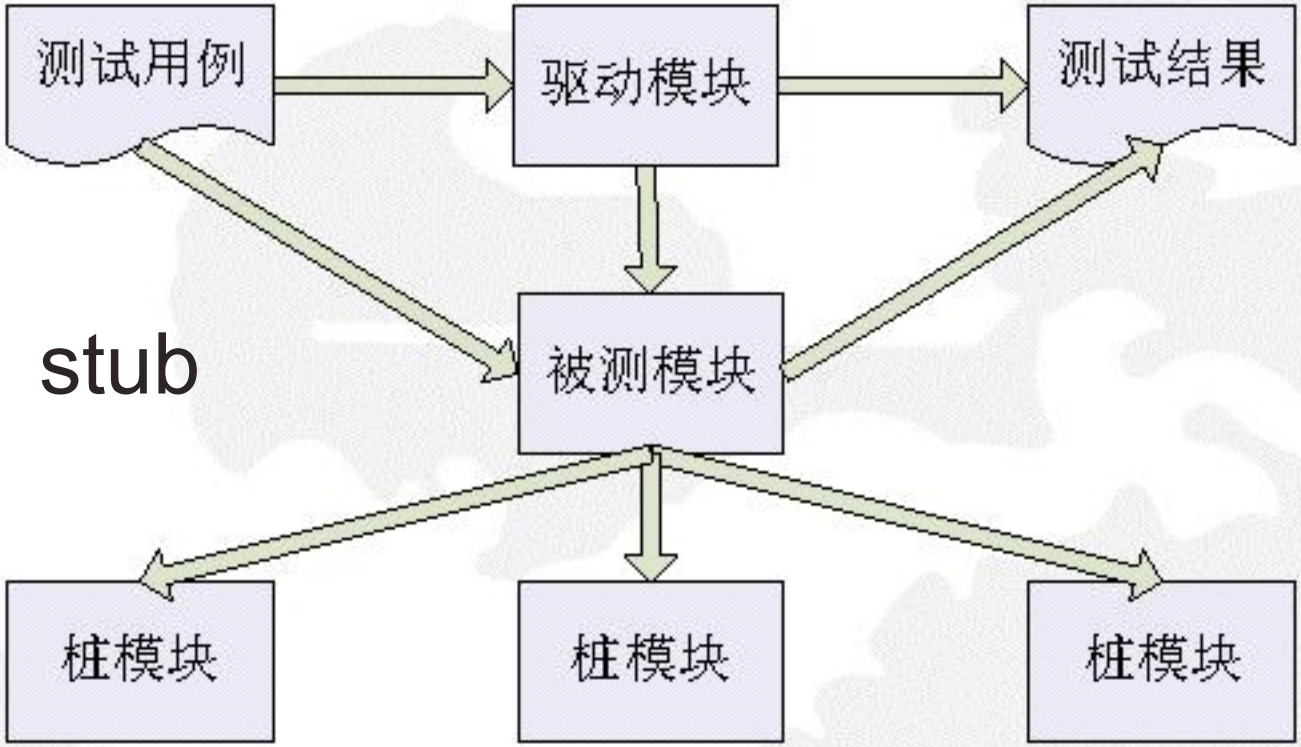
概念：

单元：有明确的功能、性能定义、接口定义的软件设计最小单位——模块，概念已经扩展为组件了。

单元测试的内容



单元测试示意图



Driver，stub

Examining the Code

[Reading assignment: Chapter 6, pp. 91-104]

Static white-box testing

- *Static white-box testing is the process of carefully and methodically reviewing the software design, architecture, or code for bugs without executing it.*

Essential elements of a formal code review

- *Identify problems:*
  - *Find problems with the software such as missing items, mistakes, etc.*
- *Follow rules:*
  - *Amount of code to be reviewed, how much time will be spent, etc.*
- *Prepare:*
  - *Each participant should prepare in order to contribute to the review.*
- *Write a report:*
  - *Summarize the results of the review, make report available to the development team.*

Informal 非正式code inspections

- ***Peer reviews*同级审查:**
  - *An informal small group of programmers and/or testers act as reviewers.*
  - *Participants should follow the 4 essential elements even through the review is informal.*
- ***Walkthroughs*:走查**
  - *A more formal process in which the author of the code formally presents the code to a small group of programmers and/or testers.*
  - *The author reads the code line by line explaining what it does, reviewers listen and ask questions.*
  - *Participants should follow the 4 essential elements.*

Formal code reviews

- *A formal code review is the process under which static white-box testing is performed.*

Formal code inspections

- Code presenter *is not the author of the code.*
- *The other participants are the* inspectors.
- *There is a* moderator *to assure that the rules are followed and the meeting runs smoothly.*
- *After the inspection a report is composed. The programmer then makes changes and a re-inspection occurs, if necessary.*

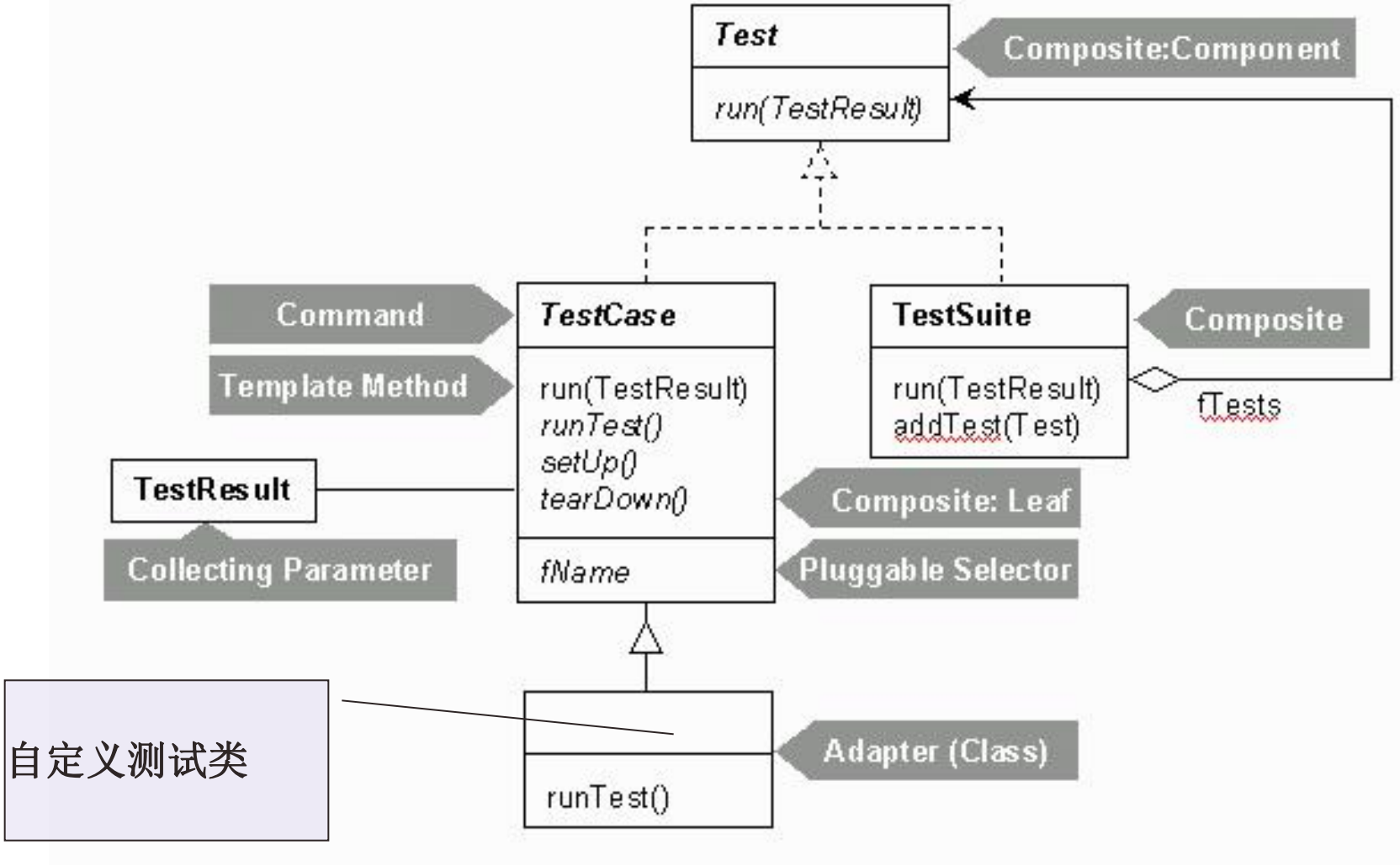
Code review checklist:

- *Data reference errors*
- *Data declaration errors*
- *Computation errors*
- *Comparison errors*
- *Control flow errors*
- *Subroutine parameter errors*
- *Input/Output errors*

xUnit框架

xUnit 框架概念		描述
测试	TestMethod	简单说，这些是您的测试。测试预期结果的逻辑，并报告未取得结果（如果有）。请将它看作您的“方法”。
测试装置	TestClass	针对大量测试的一个逻辑分组。请将它看作您的“类”。
测试套件	测试列表 **	针对大量测试装置的一个逻辑分组。请将它看作您的“类库”。 **注不需要一个属性。**
测试运行器		

JUnit框架


Test
run(TestResult)
Composite:Component
Command
Template Method
TestCase
run(TestResult)
runTest()
setUp()
tearDown()
fName
TestSuite
run(TestResult)
addTest(Test)
Composite
fTests
TestResult
Collecting Parameter
Composite: Leaf
Pluggable Selector
runTest()
Adapter (Class)
自定义测试类

JUnit *(JUnit3)*

- JUnit常用的接口和类
  - 1、Test接口：运行测试和收集测试结果；
  - 2、TestCase抽象类：定义测试中固定方法；
    - TestCase是Test接口的抽象实现（不能被实例化，只能被继承）
  - 3、Assert静态类：一系列断言方法的集合；
  - 4、TestSuite测试包类：多个测试的组合
    - TestSuite类负责组装多个TestCase,待测得类中可能包括对被测类的多个测试
    - 5、setUp方法和tearDown方法

JUnit4

- *JUnit4 使用 Java 5 中的注解（**annotation**），以下是JUnit4 常用的几个annotation介绍*
  - *@ Before：初始化方法,每开始一个新的测试都要执行一次，setup()*
  - *@After：释放资源，每完成一个测试都要执行一次， tearDown(),*
  - *@Test：测试方法，对应：testAdd()*
  - *@Ignore：忽略的测试方法*

JUnit4

- *@BeforeClass：针对所有测试，只执行一次，且必须为static void*
  @AfterClass：针对所有测试，只执行一次，且必须为static void

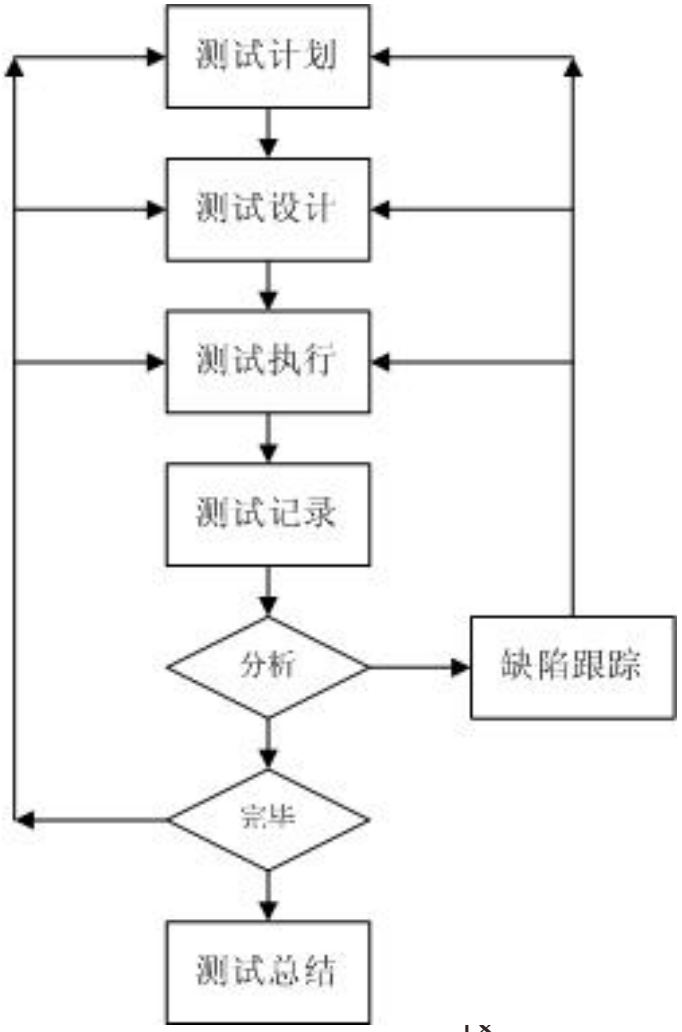
单元测试策略

- 什么时候测试？XP开发理论讲究TDD，即测试驱动开发，先编写测试代码，再进行开发。
  - 先编写产品的框架,是指先编写类、方法空的实现，需要确定接口
  - 编译通过然后编写测试类，针对产品类的功能编写测试用例，这时方法名、参数表、返回类型都应该确定下来了
  - 然后编写产品类的代码，每写一个功能点都运行测试，随时补充测试用例。
  - 编写的测试代码以后需修改的可能性比较小。

单元测试的过程和文档管理

过程:

1. 在详细设计阶段完成单元测试计划。
2. 建立单元测试环境，完成测试设计和开发。
3. 执行单元测试用例，并且详细记录测试结果。
4. 判定测试用例是否通过。
5. 提交《单元测试报告》。



单元测试工具

- JUnit

单元测试工具

- Parasoft Jtest
  是自动化Java单元测试工具。
  _Parasoft C++Test

You now know …

- *… static white-box testing*
- *… code reviews*
- *… informal code inspections*
- *… formal code inspections*
- *… code review checklists*
- *xUnit，JUnit,CppUnit*
- *单元测试工具*

1. Integration Testing

集成测试的概念

集成测试：将单元组装起来再进行测试，以检查这些单元之间的接口是否存在问题，如：数据丢失、模块间相互影响、组合后不能实现主功能等

软件集成测试前的准备

Integration Strategies

- *Top-down*
- *Bottom-up*
- *混合策略*
- *Big bang*
- *Critical-first关键先行*
- *Function-at-a-time*

集成测试的模式

渐增式测试模式与非渐增式测试模式

非渐增式测试模式：先分别测试每个模块，再把所有模块按设计要求放在一起结合成所要的程序，如大棒模式。

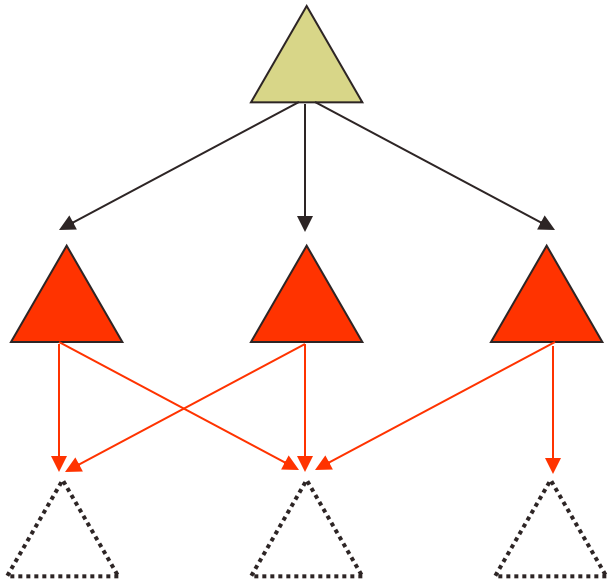
渐增式测试模式：把下一个要测试的模块同已经测试好的模块结合起来进行测试，测试完以后再把下一个应该测试的模块结合进来测试。

Big Bang Integration

- *Wait until all the components are ready, then put everything together at once*
- *Issues:*
  - *avoids cost of scaffolding (stubs or drivers)*
  - *does not provide any locality for finding faults不易发现错误*

Top-Down Integration

- *Start with top-level modules*
- *Use stubs for lower-level modules*
- *As each level is completed, replace stubs with next level of modules*

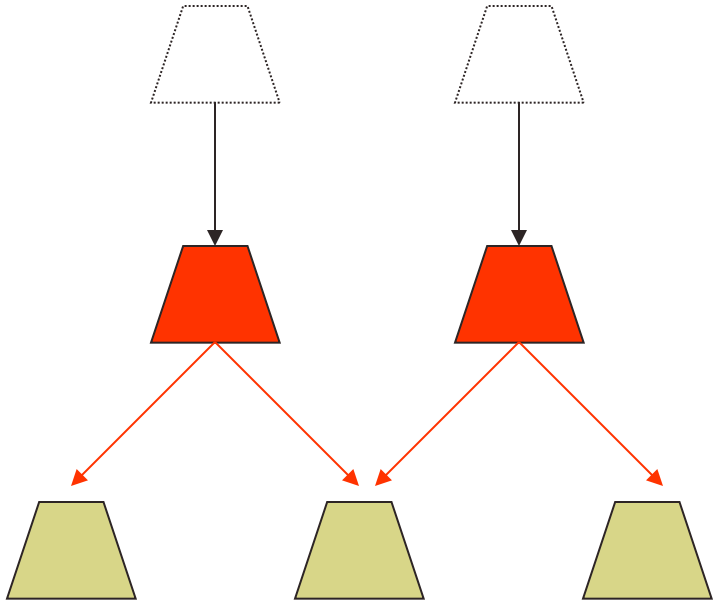


Top-Down Issues

- *Pros:*
  - *Always have a top-level system*
  - *Stubs can be written from interface specifications*
- *Cons:*
  - *May delay performance problems until too late*
  - *Stubs can be expensive*

Bottom-Up Integration

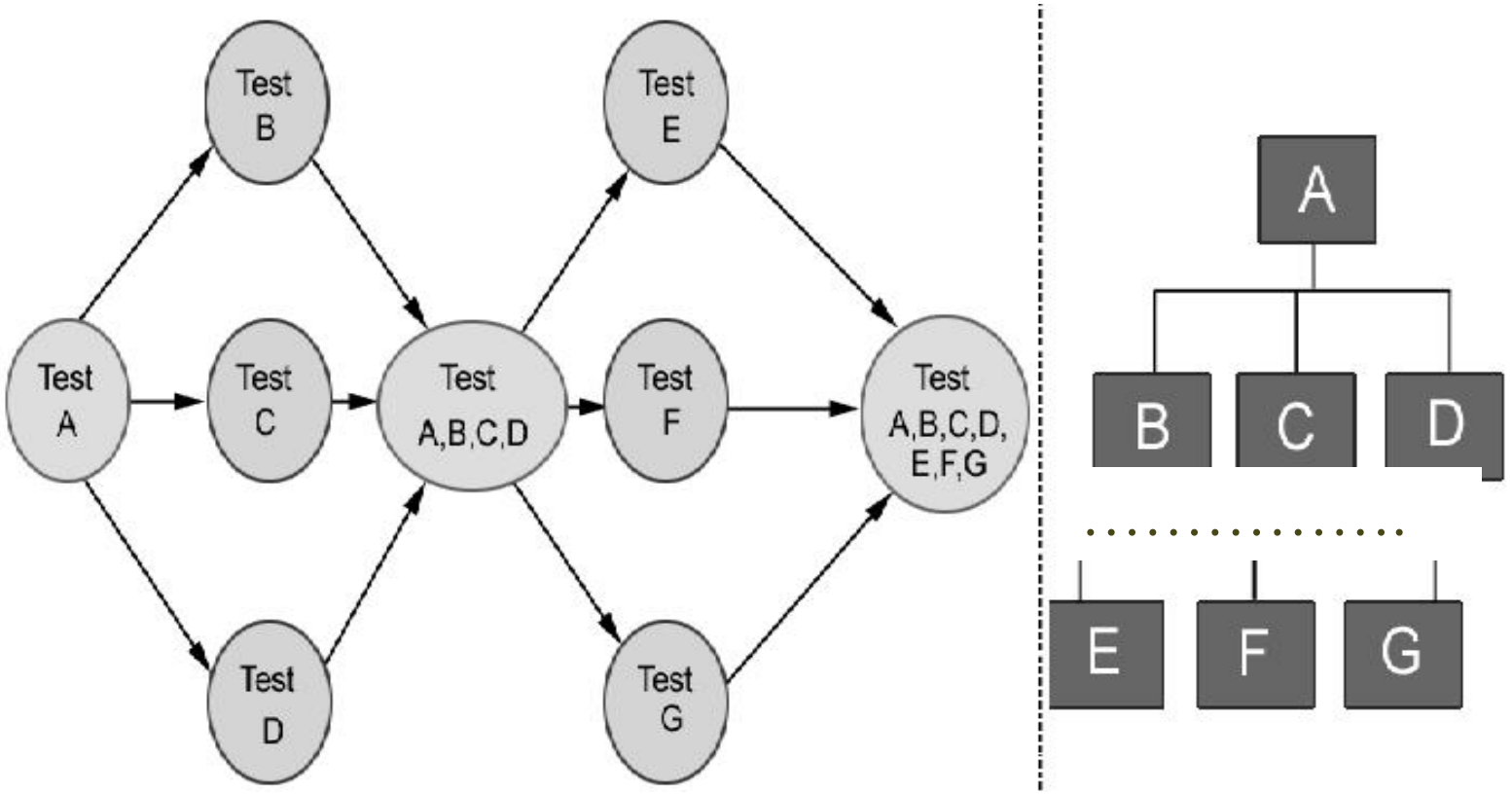
- *Start with bottom-level modules*
- *Use drivers for upper-level modules*
- *As each level is completed, replace drivers with next level of modules*



Bottom-Up Issues

- *Pros:*
  - *Primitive functions get most testing*
  - *Drivers are usually cheap*
- *Cons:*
  - *Only have a complete system at the end*

混合策略(Modified Top-down Integration)



混合法：对软件结构中较上层，使用的是“自顶向下”法；对软件结构中较下层，使用的是“自底向上”法，两者相结合

几种集成测试方法性能的比较

	自底向上	自顶向下	混合策略	大棒
集成	早	早	早	晚
基本程序能工作时间	晚	早	早	晚
需要驱动程序	是	否	是	否
需要桩程序	否	是	是	否
计划与控制	容易	难	中	容易

软件测试与维护

测试环境的建立

测试环境的五要素

测试环境的基本要素是：软件、硬件。

在基本要素的基础上派生出:网络环境、数据准备、测试工具三要素

2. System Testing

功能测试

- *在集成测试结束之后，依据系统的需求规格说明书和产品功能说明书对系统的整体功能进行的全面测试，称为功能测试*
- *采用黑盒测试方法*

功能测试的目的和内容

- 程序安装、启动正常，有相应的提示框、错误提示等
- 每项功能符合需求
- 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

功能测试的目的和内容

- 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
- 软件升级后，能继续支持旧版本的数据
- 与外部应用系统的接口有效

回归测试

- 定义：
  - 对修正缺陷后的软件进行再次的测试，不仅测试被修复的软件缺陷是否已经解决，还要测试软件旧有的功能与非功能是否满足要求

回归测试的方法

- 再测试全部用例
- 基于风险选择测试 ：测试关键的、重要的、可疑的，跳过非关键的、稳定的
- 基于操作剖面选择测试
- 再测试修改的部分

冒烟测试

- *具体说，冒烟测试就是在每日build建立后，对系统的基本功能进行简单的测试。这种测试强调功能的覆盖率。*

  简单的说，就是先保证系统能跑的起来，先通过最基本的测试，如果最基本的测试都有问题，就直接打回开发部了，减少测试部门时间的浪费

2. System Testing

Non-functional test

- 压力测试(Stress test)
- 容量测试(Capacity test)
- 性能测试(Performance test)
- 安全测试(Security test)
- 容错测试(Recovery test)

压力测试

压力测试是在一种需要反常数量、频率或资源的方式下，执行可重复的负载测试或强度测试，以检查程序对异常情况的抵抗能力，找出性能瓶颈。

压力测试-稳定性压力测试

- 稳定性压力测试，也叫可靠性测试（reliability testing），或疲劳测试，是指连续运行被测系统，检查系统运行时的稳定程度。
  - 我们通常用mtbf（mean time between failure，即错误发生的平均时间间隔）来衡量系统的稳定性，mtbf越大，系统的稳定性越强

压力测试-稳定性压力测试

- 在选定压力下，系统持续运行24小时来检测稳定性情况，检测系统的必要性能指标
- 稳定性压力测试的使用场合：为了给客户一个使用边界条件，目的是为了稳定（系统的设计性能）

压力测试-破坏性压力测试

- 破坏性压力测试：通常是指持续不断的给被测系统增加压力，直到将被测系统压垮为止（让问题与薄弱环节快速暴露出来，找出瓶颈）。 用来测试系统所能承受的最大压力。

容量测试

容量测试目的是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**（如最大并发用户数、数据库记录数等），系统在其极限值状态下还能保持主要功能**正常运行**。容量测试还将确定测试对象在**给定时间**内能够**持续处理**的**最大负载或工作量**。

通过**压力测试的稳定性测试**，可以得到**容量值**的。

性能测试

性能测试：

•通过测试确定系统运行期间的**性能表现与性能数据**，得到如：CPU使用的效率、运行速度、响应时间、占有系统资源等方面的系统数据

作用：

•确定在什么样的压力下系统达到最佳状态（稳定性压力测试），什么压力是系统的极限（破坏性压力测试）

•根据性能指标确定实际的软硬件运行环境：如配置合理的cpu数、cpu的处理能力、内存量等等

性能测试与瓶颈分析关键步骤

- 步骤一：性能测试与数据收集
- 步骤二：性能瓶颈分析
- 步骤三：性能调优解决方案

压力测试工具－LoadRunner

- 1）Virtual User Generator 创建脚本
  - 创建脚本，选择协议
  - 录制脚本
  - 编辑脚本
  - 检查修改脚本是否有误
- 2）中央控制器（Controller）来调度虚拟用户
  - 创建Scenario，选择脚本
  - 设置机器虚拟用户数
  - 设置Schedule
  - 如果模拟多机测试，设置Ip Spoofer
- 3）运行脚本
  - 分析scenario
- 4）分析测试结果

安全性测试，可靠性和容错性测试

安全性测试、可靠性测试和容错性测试的测试目的不同，其手段和方法也不同，但都属于系统测试的范畴，有一定的联系，如软件可靠性要求通常包括了安全性的要求。而且，安全性测试、可靠性测试和容错性测试的技术比较深、实施比较难，但在计算机应用系统中其作用越来越大。

安全性测试

安全性测试是检查系统对非法侵入的防范能力。安全测试期间，测试人员假扮非法入侵者，采用各种办法试图突破防线。

系统安全设计的准则是，使非法侵入的代价超过被保护信息的价值，此时非法侵入者已无利可图

容错性测试

容错性测试是检查软件在异常条件下自身是否具有**防护性的措施**或者某种**灾难性恢复的手段**。如当系统出错时，能否在指定时间间隔内修正错误并重新启动系统。

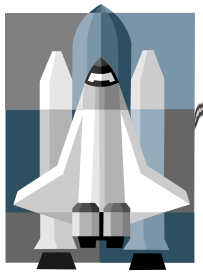
Acceptance Testing

验收测试

验收测试(Acceptance Test):在软件产品完成了功能测试和系统测试之后、产品发布之前所进行的软件测试活动

它是技术测试的最后一个阶段,也称为交付测试。

验收标准和注意事项



验收测试完成标准：

- 完全执行了验收测试计划中的每个测试用例。
- 在验收测试中发现的错误已经得到修改并且通过了测试或者经过评估留待下一版本中修改。
- 完成软件验收测试报告。

注意事项：

- 必须编写正式的、单独的验收测试报告
- 验收测试必须在实际用户运行环境中进行
- 由用户和测试部门共同执行。如公司自开发产品，应由测试人员，产品设计部门，市场部门等共同进行。

Other Paths to Acceptance

α 测试是指软件开发公司组织内部人员模拟各类用户对即将面市软件产品（称为 α 版本）进行测试，试图发现错误并修正。

经过 α 测试调整的软件产品称为 β 版本。紧随其后的 β 测试是指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，并报告异常情况、提出批评意见。然后软件开发公司再对 β 版本进行改错和完善。

用户界面和可用性测试

用户界面的7个要素：

- 符合标准和规范。
- 直观性。
- 一致性。
- 灵活性。
- 舒适性。
- 正确性。
- 实用性。

易用性测试没有具体量化的指标，主观性较强。

向前和向后兼容

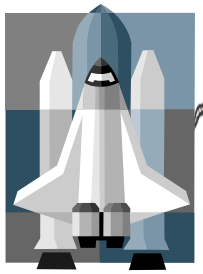
- 向后兼容是指可以使用软件的以前版本。
- 向前兼容指的是可以使用软件的未来版本。

7.5 可安装性和可恢复性测试

可安装性测试:

- 系统软件安装
- 应用软件安装
- 服务器的安装
- 客户端的安装
- 产品升级安装
- 等等

怎样进行文档测试



好的文档能达到提高易用性、提高可靠性、降低技术支持的费用的目的，从而提高了产品的整体质量。

软件测试与维护

报告所发现的软件缺陷与测试报告

软件缺陷的基本描述

- 软件缺陷的描述是软件缺陷报告中测试人员对问题的陈述的一部分并且是软件缺陷报告的基础部分。同时，软件缺陷的描述也是测试人员就一个软件问题与开发小组交流的最初且最好的机会。
- 一个好的描述，需要使用简单的、准确的、专业的语言来抓住缺陷的本质。

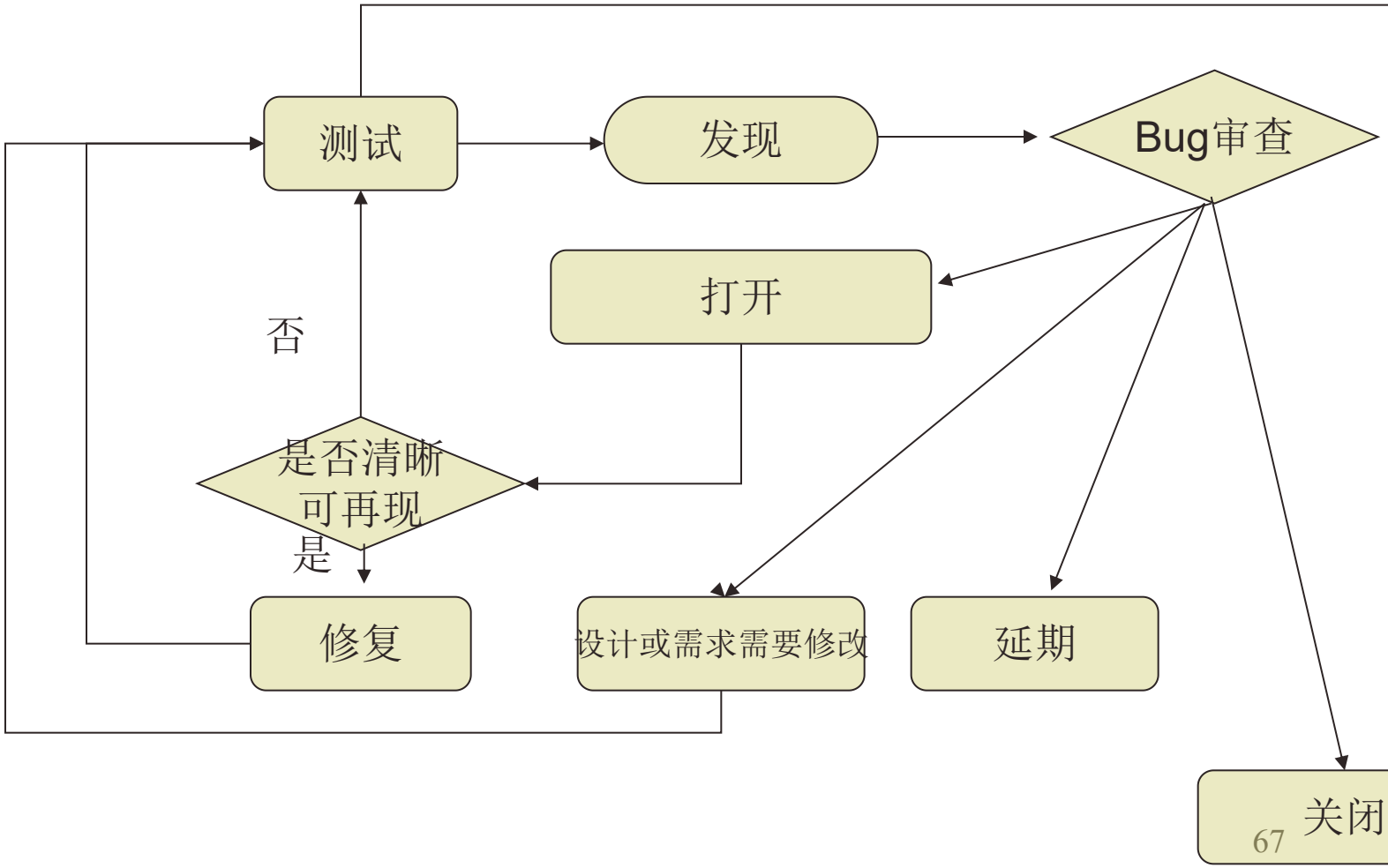
软件缺陷属性

- 软件缺陷属性包括
  - 缺陷标识
  - 缺陷类型
  - 缺陷严重程度
  - 缺陷优先级
  - 缺陷产生可能性
  - 缺陷状态
  - 缺陷起源
  - 缺陷来源
  - 缺陷原因

软件缺陷相关的信息

- 保证开发人员和其他的测试人员可以分离和重现它。
- 软件缺陷的图片、记录信息
  - 记录软件缺陷的相关图片

复杂的软件缺陷生命周期



缺陷管理工具

- *常见免费缺陷管理工具:*
  Buggit
  Mantis
  Bugzilla

软件缺陷报告

任何一个缺陷跟踪系统的核心都是“软件缺陷报告”，一份软件缺陷报告详细信息如表

软件缺陷项目列表

分类	项目	描述
可跟踪信息	缺陷ID	唯一的、自动产生的缺陷ID，用于识别、跟踪、查询
软件缺陷基本信息	缺陷状态	可分为“打开或激活的”、“已修正”、“关闭”等
	缺陷标题	描述缺陷的最主要信息
	缺陷的严重程度	一般分为“致命”、“严重”、“一般”、“较小”等四种程度
	缺陷的优先级	描述处理缺陷的紧急程度，1是优先级最高的等级，2是正常的，3是优先级最低的
	缺陷的产生频率	描述缺陷发生的可能性1%-100%
	缺陷提交人	缺陷提交人的名字（会和邮件地址联系起来），一般就是发现缺陷的测试人员或其他人员
	缺陷提交时间	缺陷提交的时间

软件缺陷报告

软件缺陷基本信息	缺陷所属项目/模块	缺陷所属的项目和模块，最好能较精确的定位至模块
	缺陷指定解决人	估计修复这个缺陷的开发人员，在缺陷状态下由开发组长指定相关的开发人员；也会自动和该开发人员的邮件地址联系起来，并自动发出邮件
	缺陷指定解决时间	开发管理员指定的开发人员修改此缺陷的时间
	缺陷验证人	验证缺陷是否真正被修复的测试人员；也会和邮件地址联系起来
	缺陷验证结果描述	对验证结果的描述（通过、不通过）
	缺陷验证时间	对缺陷验证的时间
缺陷的详细描述	步骤	对缺陷的操作过程，按照步骤，一步一步地描述
	期望的结果	按照设计规格说明书或用户需求，在上述步骤之后，所期望的结果，即正确的结果
	实际发生的结果	程序或系统实际发生的结果，即错误的结果
测试环境说明	测试环境	对测试环境描述，包括操作系统、浏览器、网络带宽、通讯协议等
必要的附件	图片、Log文件	对于某些文字很难表达清楚的缺陷，使用

测试报告

- 测试报告是把测试的过程和结果写成文档，并对发现的问题和缺陷进行分析，为纠正软件的存在的质量问题提供依据，同时为软件验收和交付打下基础。
- 测试报告是测试阶段最后的文档产出物，优秀的测试经理应该具备良好的文档编写能力，一份详细的测试报告包含足够的信息，包括产品质量和测试过程的评价，测试报告基于测试中

测试报告的内容

- *测试报告的内容可以总结为以下目录：*
  - *首页*
  - *引言（目的、背景、缩略语、参考文献）*
  - *测试概要(测试方法、范围、测试环境、工具)*
  - *测试结果与缺陷分析（功能、非功能）*
  - *测试结论与建议（项目概况、测试时间 测试情况、结论性能汇总）*
  - *附录（缺陷统计）*

小结

关键词:

软件缺陷属性、软件缺陷生命周期、软件缺陷报告、软件缺陷管理系统、软件缺陷统计曲线

测试报告