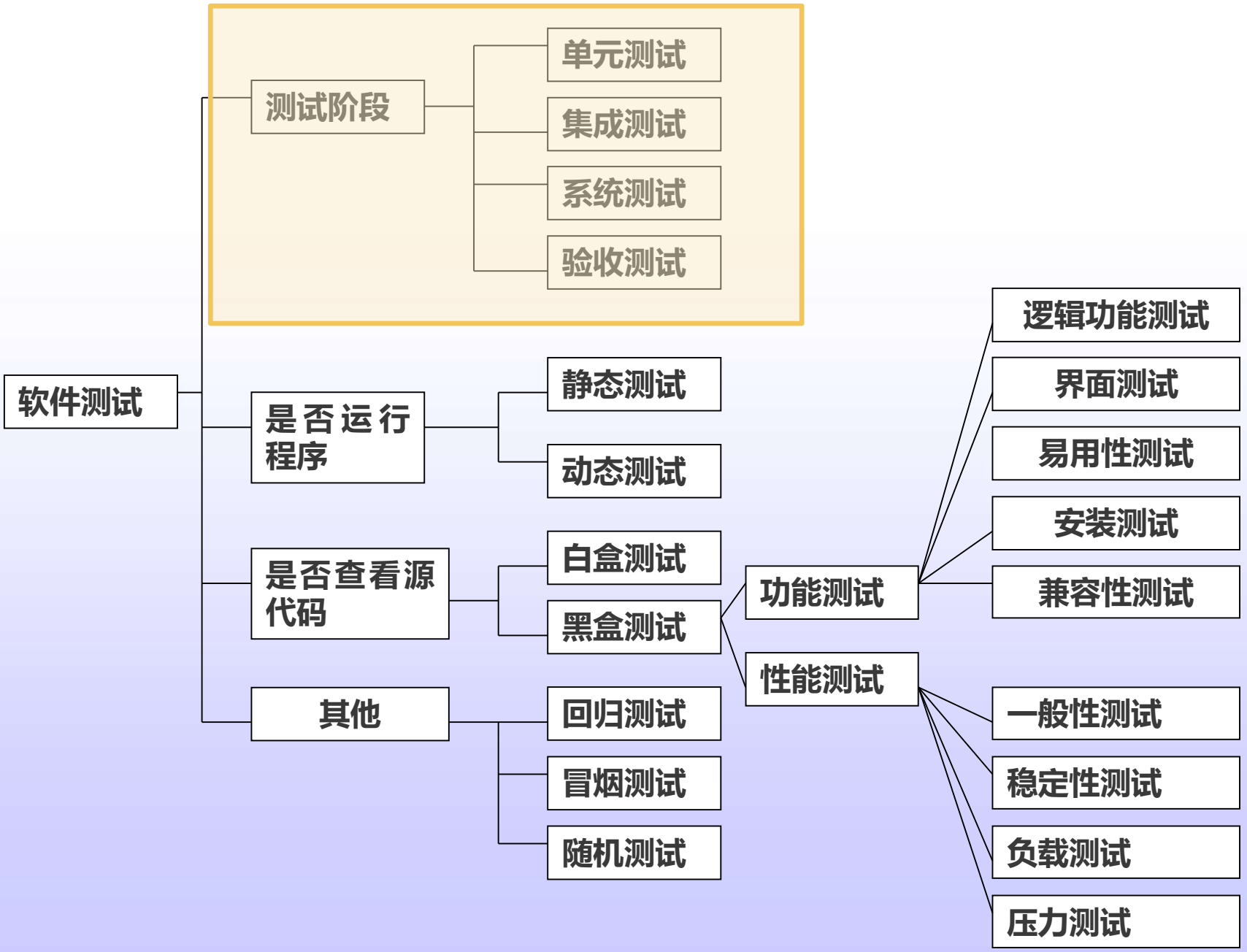

软件测试
测试阶段
单元测试
集成测试
系统测试
验收测试
是否运行程序
静态测试
动态测试
是否查看源代码
白盒测试
黑盒测试
功能测试
逻辑功能测试
界面测试
易用性测试
安装测试
兼容性测试
性能测试
一般性测试
稳定性测试
负载测试
压力测试
其他
回归测试
冒烟测试
随机测试

软件测试级别

提纲

- 软件测试级别概述
- 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）

提纲

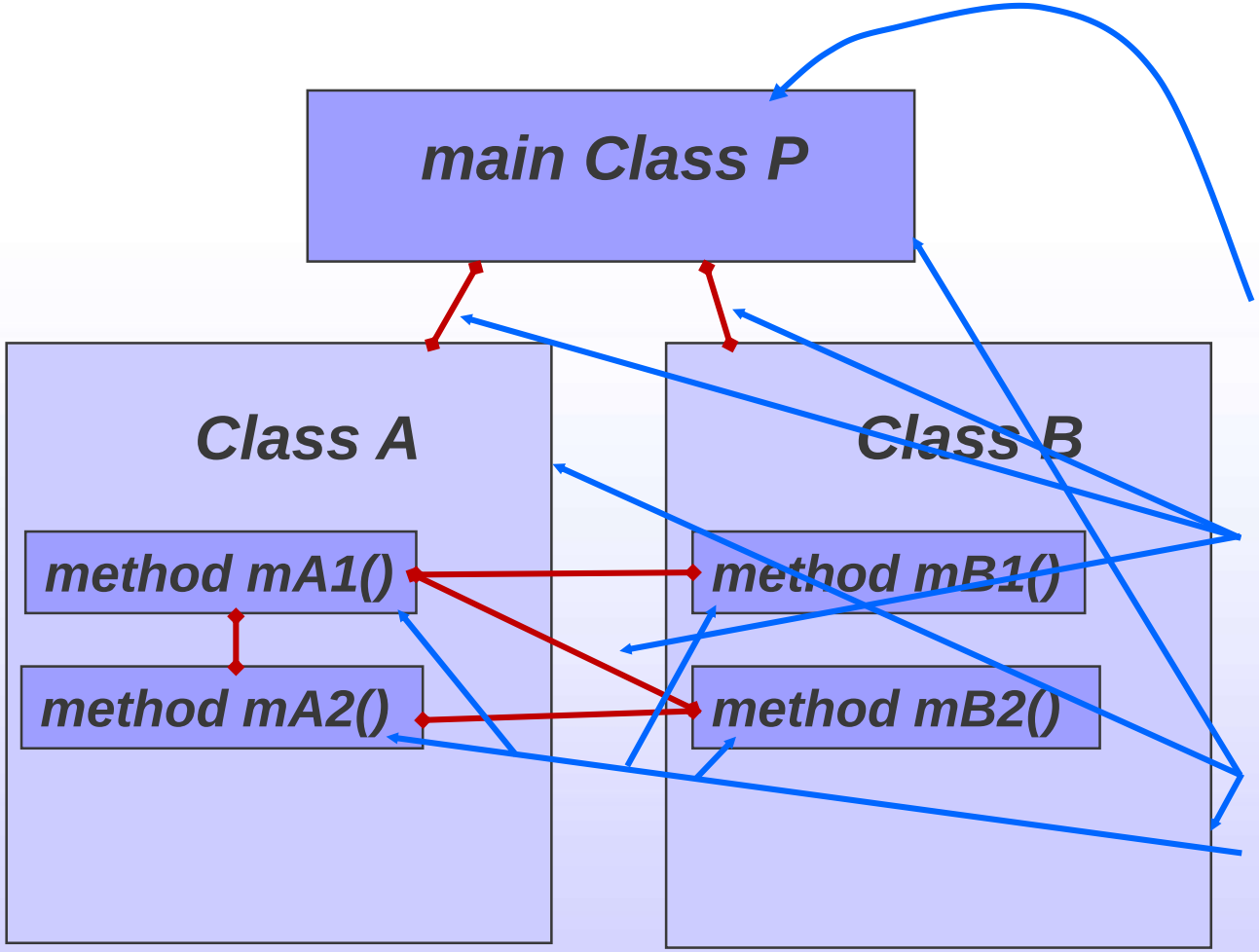
- 软件测试级别概述
- 单元测试（Unit Testing）
- 集成测试（Integration Testing ）
- 系统测试（System Testing ）
- 验收测试（Acceptance Testing ）
- 回归测试（Regression Testing）

1 软件测试级别概述

- 软件测试要经过4个不同的测试阶段，即单元测试、集成测试、系统测试和验收测试，将这些测试阶段称为软件测试级别.

按阶段进行测试是一种基本的测试策略。

软件测试级别



Acceptance testing : *Is the software acceptable to the user?*

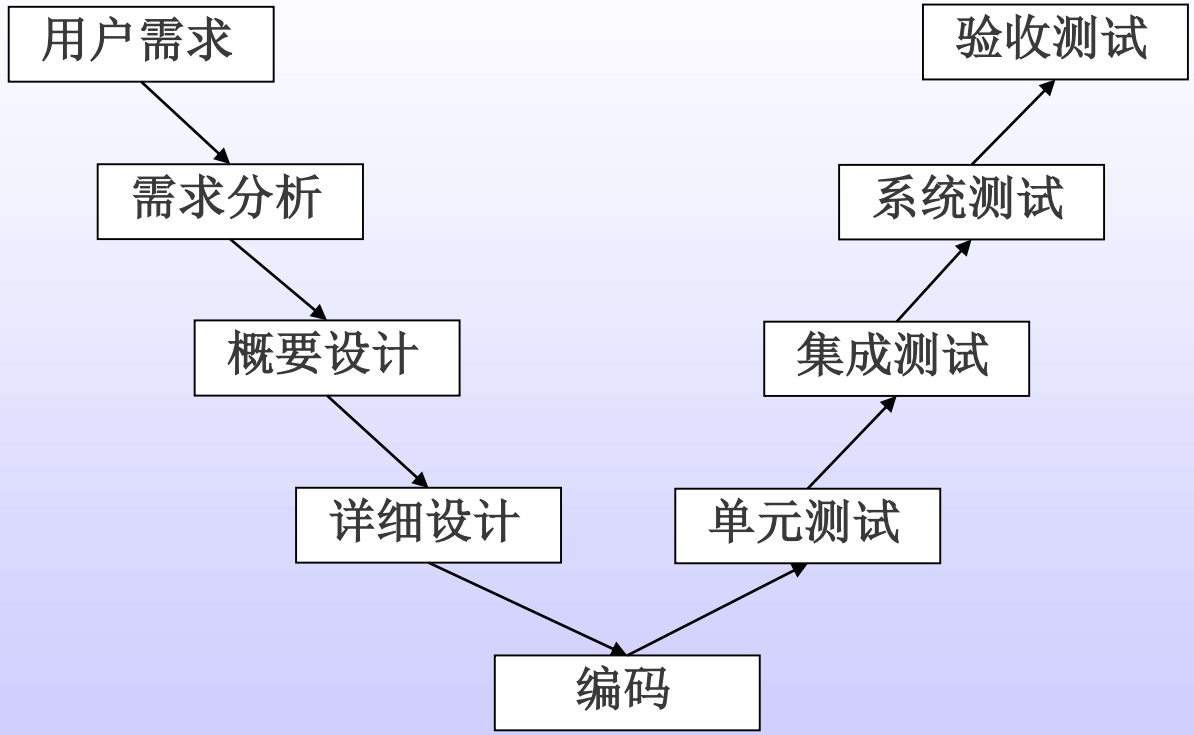
System testing :*Test the overall functionality of the system*

Integration testing :*Test how modules interact with each other*

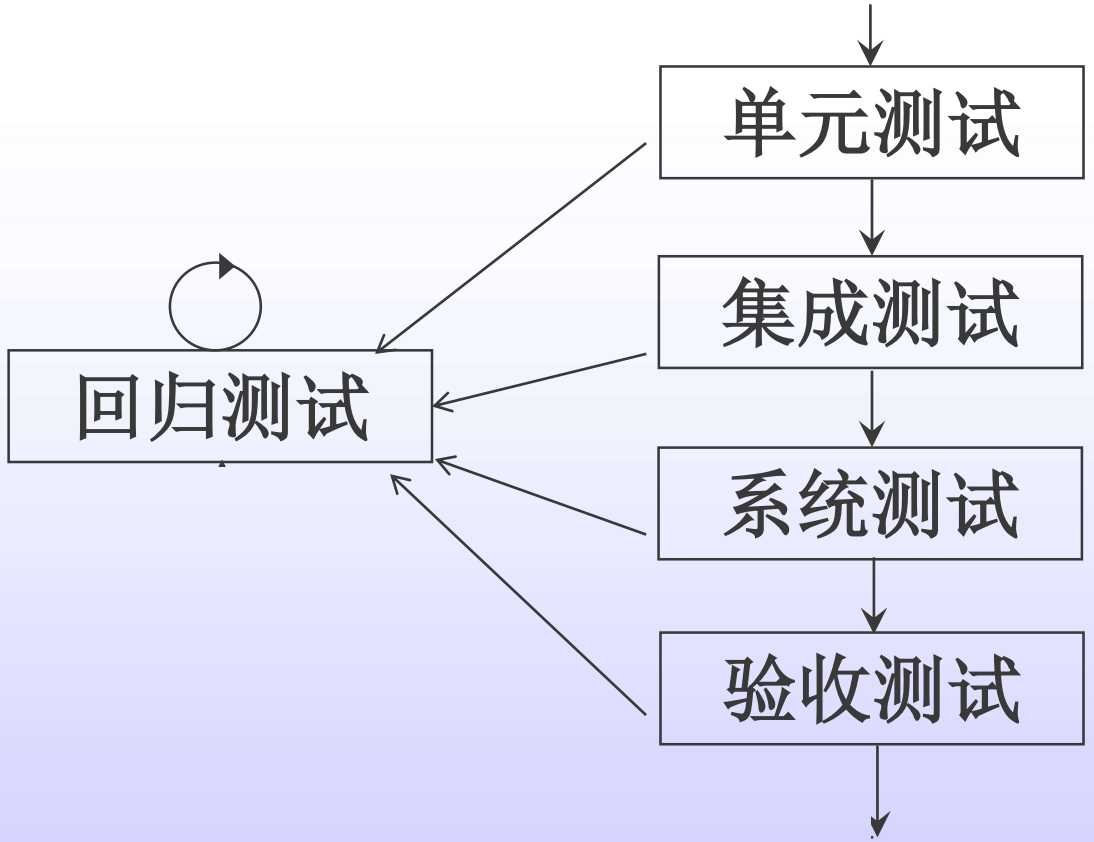
Unit testing (developer testing) :*Test each unit (method) individually,Test each class, file, module, component*

1.1 软件测试级别与开发的关系

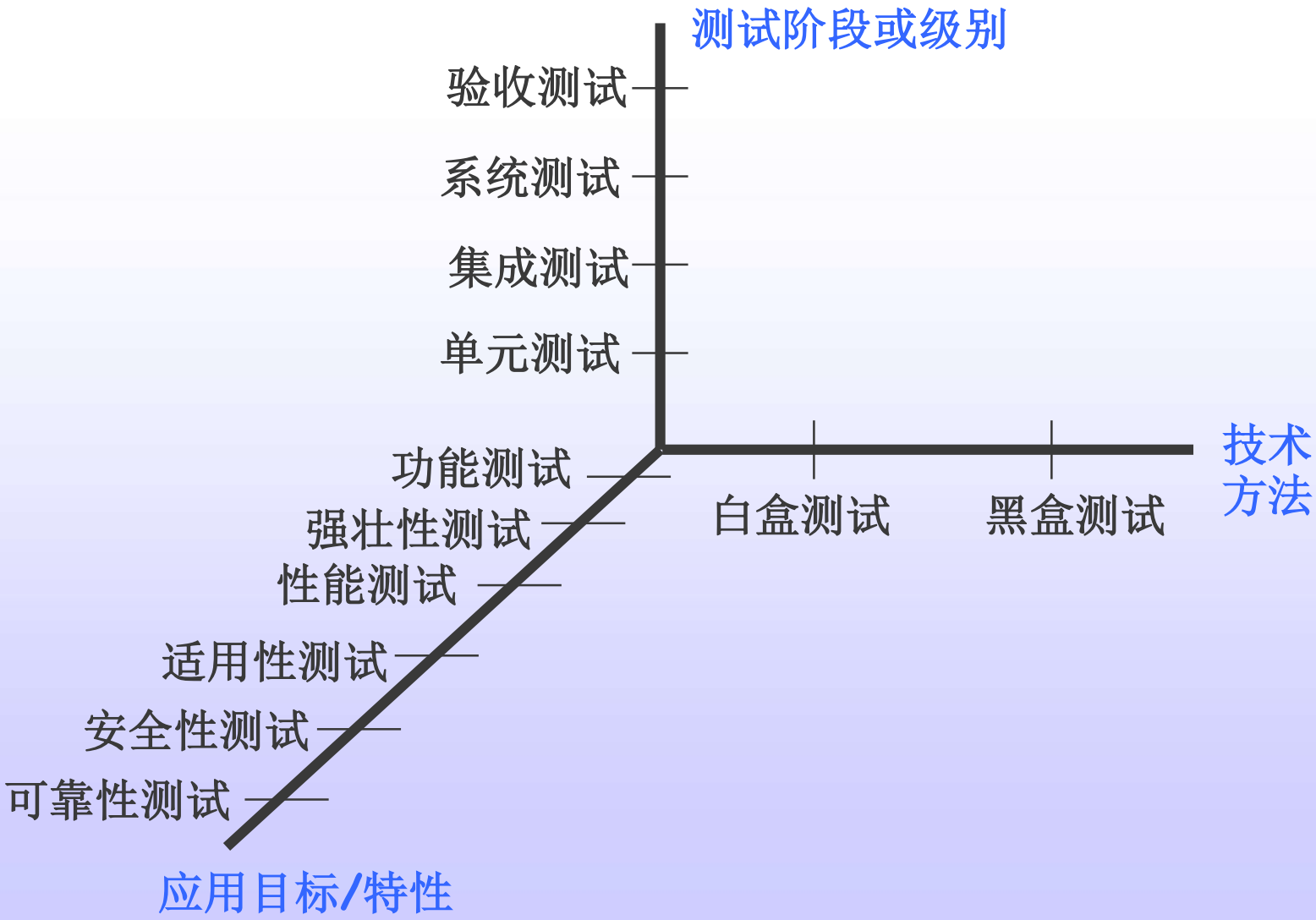
- 软件开发与软件测试级别的对应关系



1.2 各级别测试的执行顺序



1.3 软件测试级别与测试方法、测试目标的关系



提纲

- 软件测试级别概述
- 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）

2 单元测试（Unit Testing）

- 单元测试的概念
- 为什么要引入单元测试
- 什么是单元
- 单元测试的方法
- 单元测试需要的辅助手段
- 单元测试的基本任务
- 单元测试常用工具

2.1单元测试的概念

- 单元测试——是对一个模块的测试，是对软件中的基本组成单位进行的测试，如一个类、一个过程等等
- 是软件测试的最基本部分，其目的是检验软件基本组成单位的正确性
- 一般在代码完成后由开发人员完成，QA人员辅助
- 单元测试的依据是“软件详细设计”

2.2为什么要引入单元测试

你能看出它的问题吗？

2.2.1 组件的问题最终酿成惨剧

2008汶川地震，聚源中学教学楼倒塌

2.2.2 为什么要引入单元测试

- 单元测试强调的是基本的质量思想：如果要保证一个系统的质量，首先要保证构成这个系统的所有组成成分的质量
- 单元测试强调的是基础的重要作用

2.2.3 单元测试的重要性

1. 测试效果和时间的影响，认真做好单元测试后，集成测试会变得顺利，整体上更节约时间
2. 测试成本的影响，单元测试容易定位错误，而且通常在早期发现错误
3. 产品质量的影响，单元测试是构筑软件质量的基石

2.3 什么是单元

- 单元：是开发者创建的最小软件片段，单元是构造系统的基础。
- 不同的编程语言有不同的基本单元：
  - C语言的基本单元是函数；
  - C++和Java的基本单元是类；
  - Basic和COBOL的基本单元可能是整个程序

2.4 单元测试的方法

单元测试的基本方法包括:

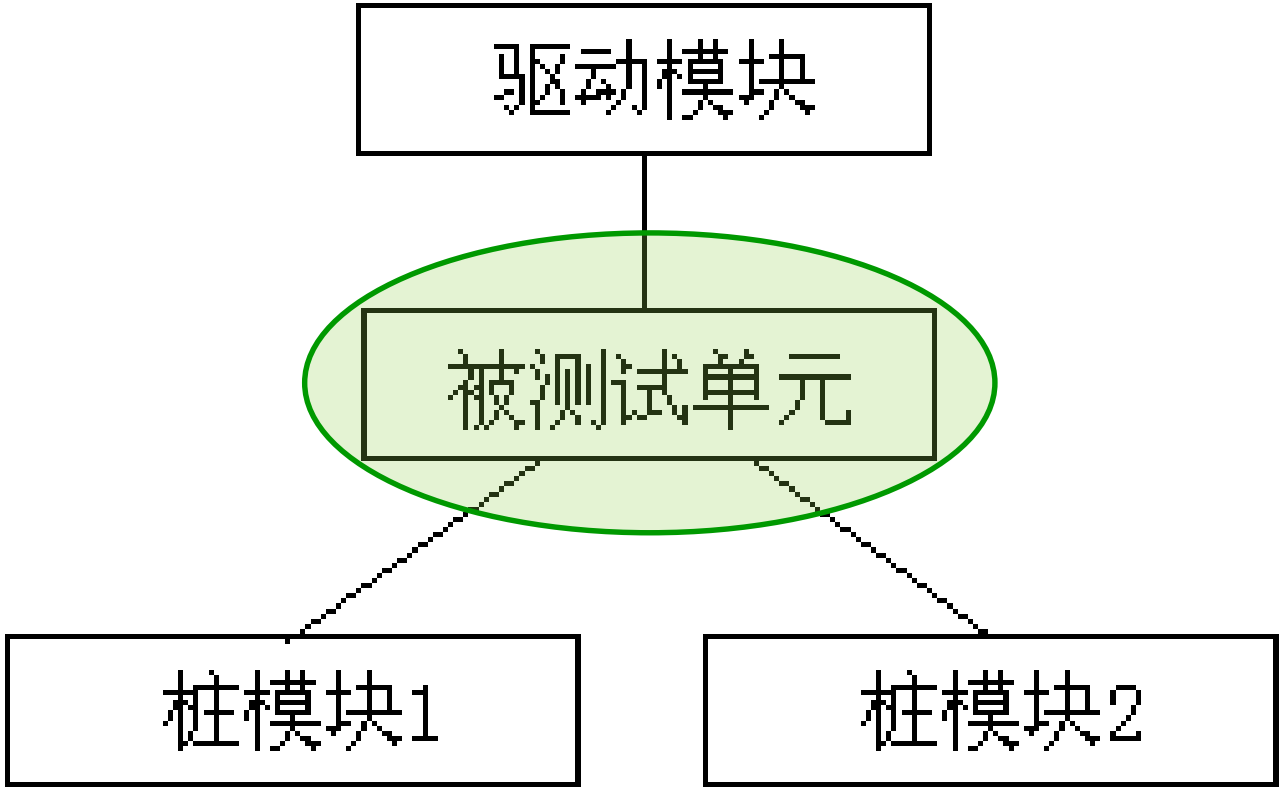
- 人工静态分析——通过人工阅读代码来查找错误，一般是程序员交叉查看对方的代码。白盒
- 自动静态分析 ——使用工具扫描代码，根据某些预先设定的错误特征，发现并报告代码中的可能错误。白盒

2.4 单元测试的方法（续）

- 人工动态测试——人工设定程序的输入和预期输出，执行程序，并判断实际输出是否符合预期，如果不符合预期，则报告错误。白盒或黑盒
- 自动动态测试——使用工具自动生成测试用例并执行被测试程序，来发现并报告错误。白盒或黑盒
- 单元测试最常使用的是白盒测试

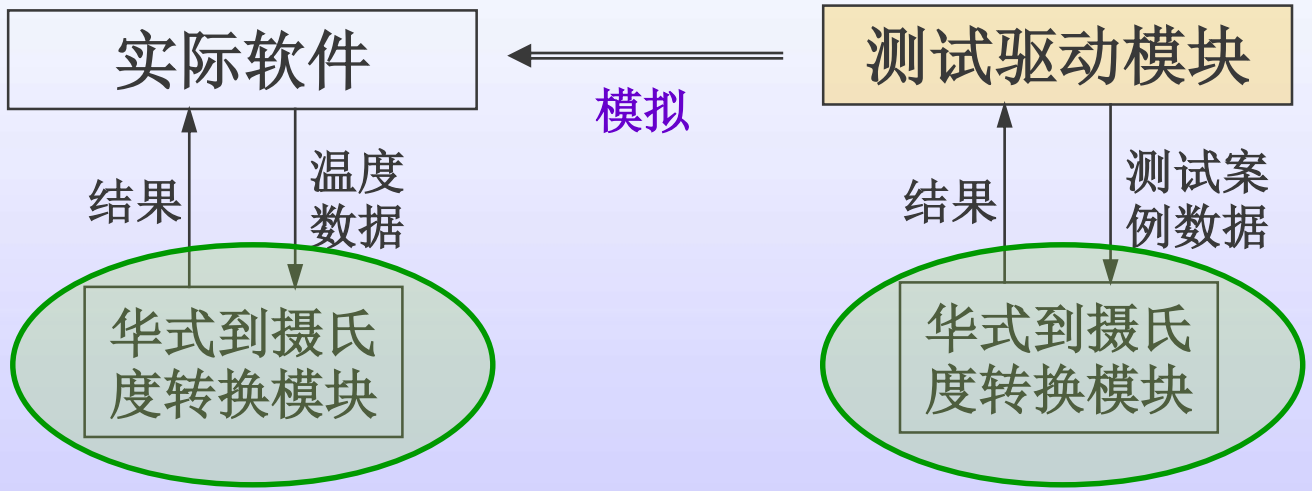
2.5 动态单元测试需要的辅助手段

- 进行动态单元测试时，因为一个被测单元无法单独运行，只有在增加了辅助代码之后才能进行单元测试
- 辅助代码包括：
  - ✓ 驱动程序/模块(Driver)
  - ✓ 桩程序/模块（Stub）



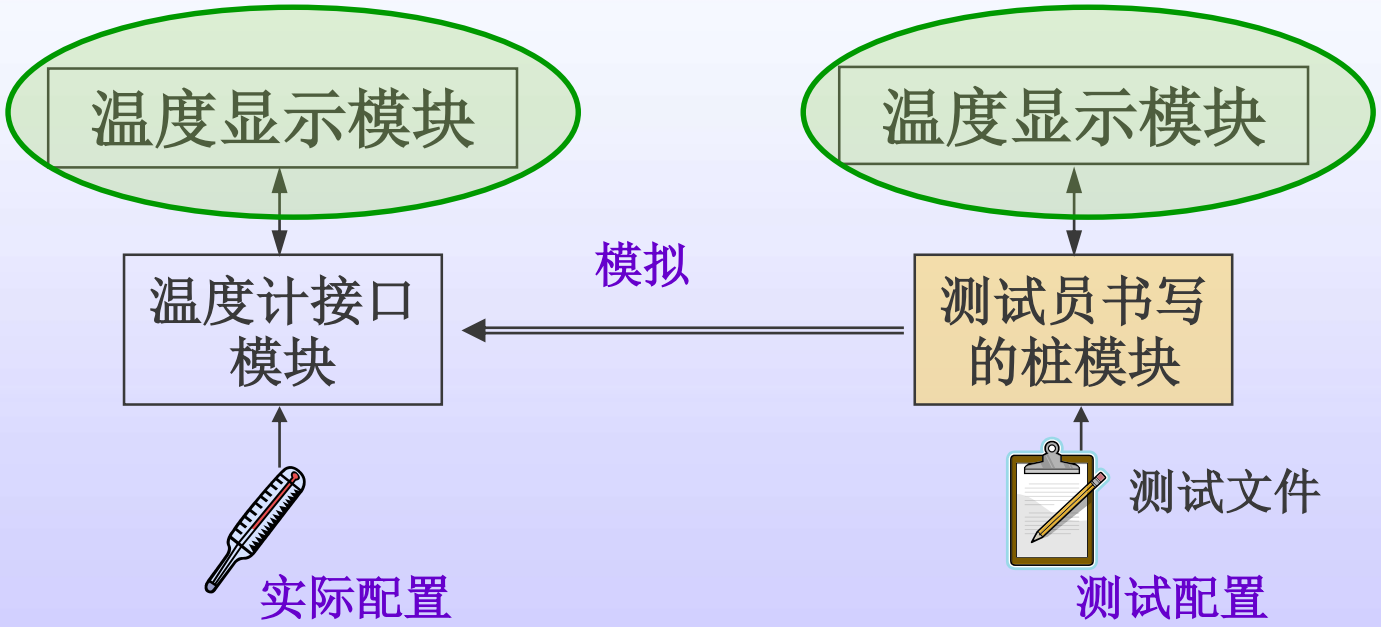
2.5.1 驱动程序/模块(Driver)

- 驱动模块（Driver）——在对底层或子层模块进行单元测试时，所编制的调用被测模块的程序，用于模拟被测模块的上级模块



2.5.2 桩模块（Stub）

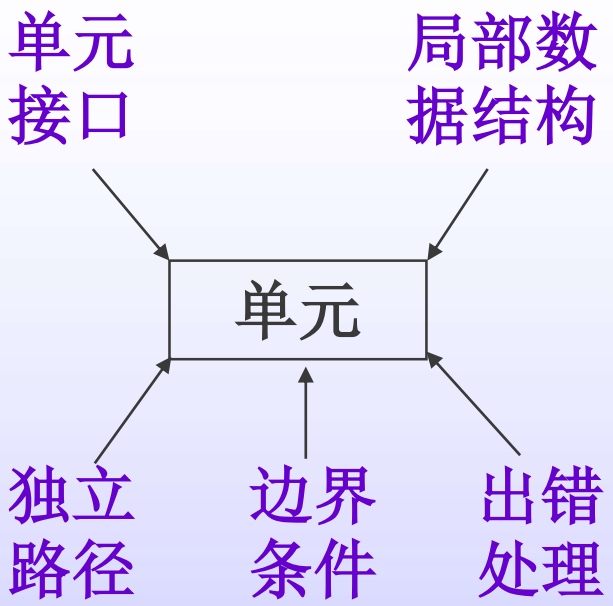
- 桩模块（Stub）——对顶层或上层模块进行单元测试时，所编制的替代下层模块的程序，用于模拟被测模块工作过程中所调用的模块



2.6 单元测试的基本任务

- 单元测试解决5个方面的测试问题：

1. 单元接口测试
2. 局部数据结构测试
3. 独立执行路径测试
4. 各种错误处理测试
5. 单元边界条件测试



与单元测试相关的测试

2.6.1 单元接口测试

- 单元接口测试检查进出模块的数据流是否正确，这是单元测试的基础

1. 参数的个数、类型是否匹配
2. 参数的顺序是否正确
3. 是否会修改只读参数
4. 参数的假设与实际是否一致
5. 全程变量的定义在各模块是否一致

2.6.2 单元局部数据结构测试

- 测试单元内部数据的完整性、正确性等

1. 不正确的或不一致的类型说明
2. 无初始值、错误的初始化或默认值
3. 错误的变量名
4. 不相容的数据类型
5. 上溢、下溢或地址错误

2.6.3 单元中所有独立路径测试

- 测试单元中每一条独立路径，可以发现控制流错误：

1. 误解的或不正确使用算符优先级
2. 混合类型运算
3. 不正确的判定
4. 不正确的逻辑操作或优先级
5. 不适当地修改循环变量
6. 不正常的或不存在的循环终止

2.6.4 各种错误处理测试

- 良好的设计应该预先估计到软件运行时可能发生的错误，并给出相应的处理措施（容错性）

1. 对运行发生的错误描述难以理解
2. 报告的错误与实际遇到的错误不一致
3. 出错后，在错误处理之前引起了系统干预
4. 异常条件的处理不正确
5. 提供的错误信息不足，以致无法定位错误

2.6.5 边界条件测试

- 检查边界数据处理的正确性，采用边界分析方法设计测试用例

1. 普通合法数据的处理
2. 普通非法数据的处理
3. 边界值内、外边界数据处理
4. 数据流、控制流中等于、大于、小于比较是否出错

2.7 单元测试常用工具

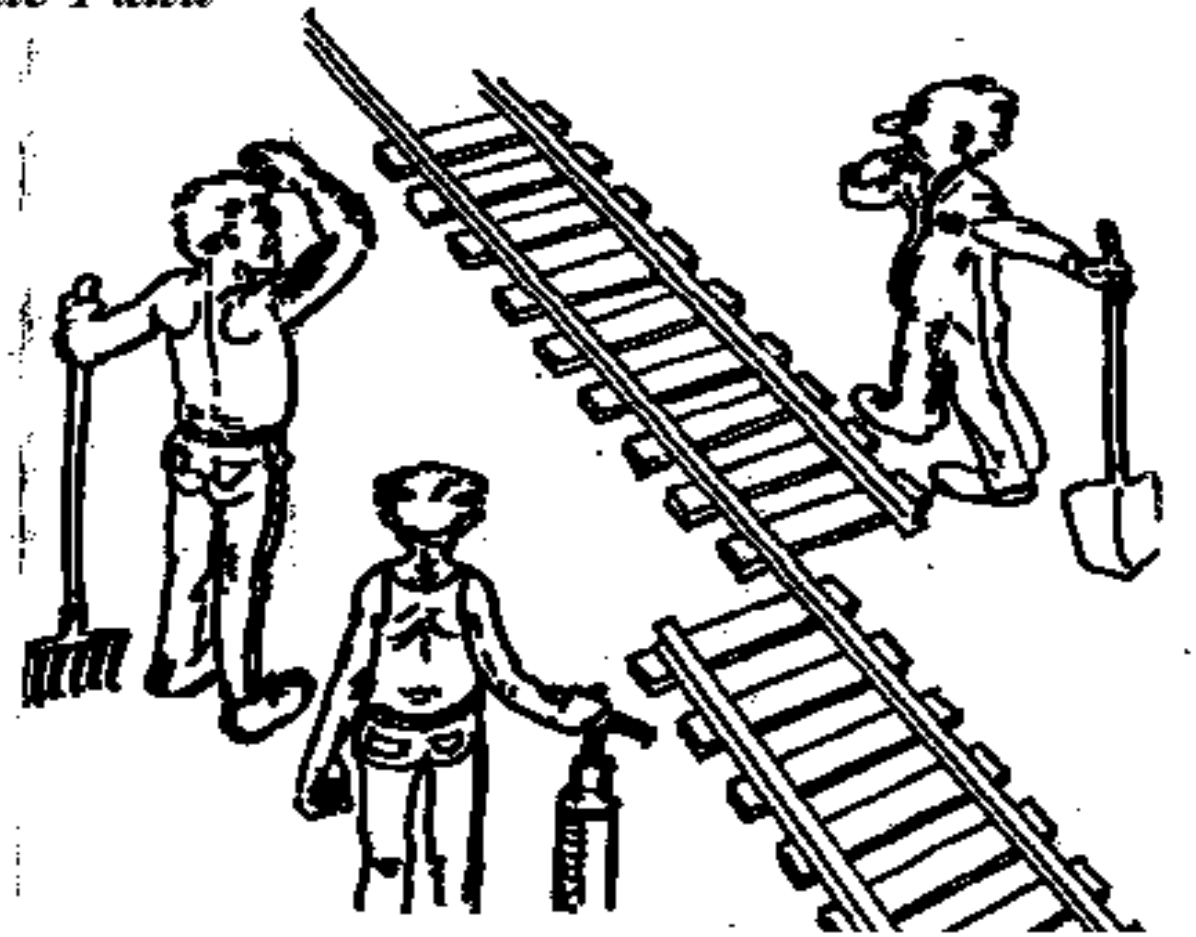
- 静态分析工具
- 代码规范审核工具
- 内存和资源检查工具
- 测试数据生成工具
- 测试框架工具 **xUnit**
- 测试结果比较工具
- 测试度量工具
- 测试文档生成和管理工具

提纲

- 软件测试级别概述
- 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）

为什么总是集成不起来？

Algorithmic Fault



3 集成测试

- 集成测试概述
- 集成测试的主要任务
- 集成测试的实施方案
- 集成测试的原则

3.1 集成测试概述

- 单元测试完成之后，将单元连接起来组成软件系统的过程叫做集成。
- 集成的过程就是形成子系统及系统的过程。对这个过程中的子系统进行测试称为集成测试。
- 集成测试的主要依据是《软件概要设计》

3.2 集成测试的主要任务

- 集成测试解决以下5个方面的测试问题

1. 将各单元连接起来，单元相互调用时，数据经过接口是否丢失
2. 将各个子功能组合起来，检查能否达到预期要求的各项功能
3. 一个单元的功能是否会对另一个单元的功能产生不利的影响
4. 全局数据结构是否存在问题，会不会被异常修改
5. 单个单元的误差积累起来，是否被放大

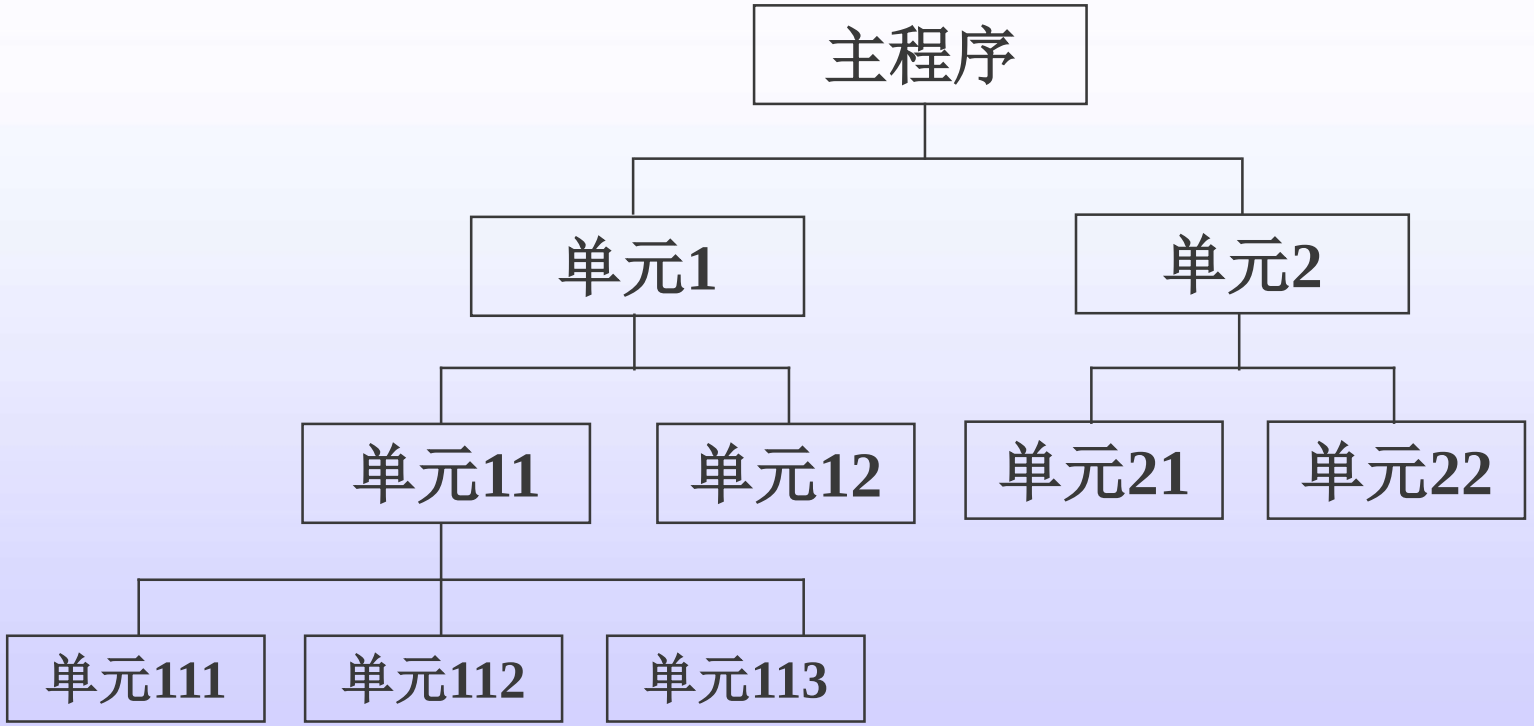
3.3 集成测试的实施方案

- 集成测试的实施方案依赖于单元的集成策略，集成策略包括：

1. 非增量式集成
2. 增量式集成

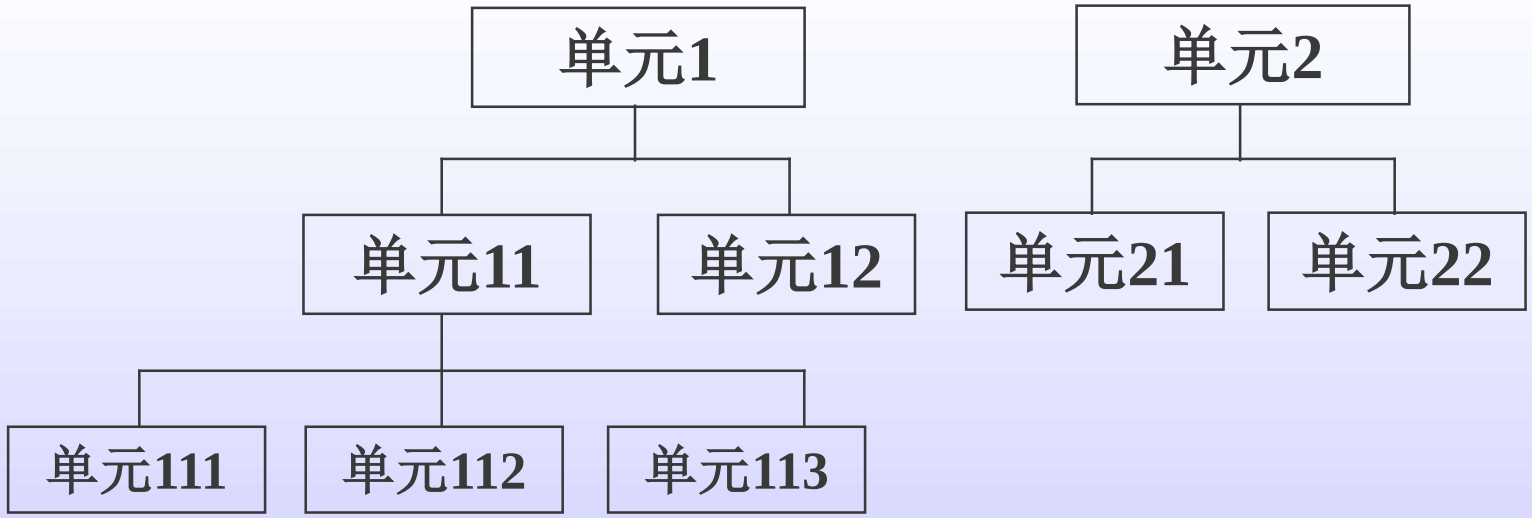
3.3.1 非增量式集成测试

- 非增量式集成测试采用一步到位的方法进行测试，即在完成单元测试后，一次性将所有单元连接起来进行整体测试

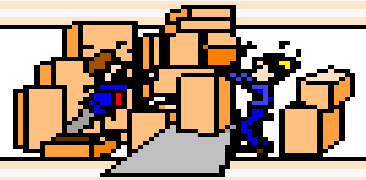


3.3.2 增量式集成测试

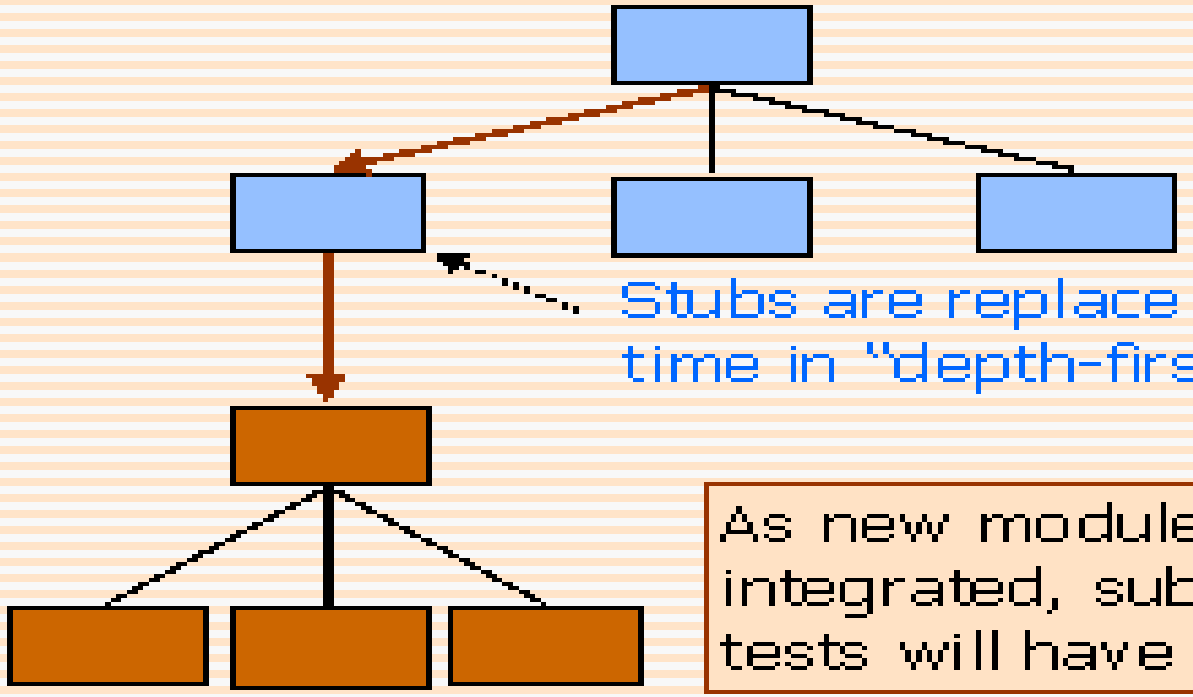
- **增量式集成：**逐步完成测试单元的集成，最后形成整个系统
- 具体实施策略：自顶向下、自底向上、混合式



自顶向下集成



Top-down Integration

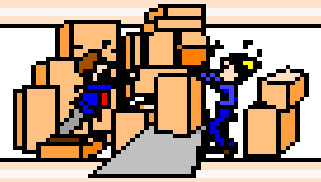


Higher-level modules (or system initiation classes) are tested with stubs

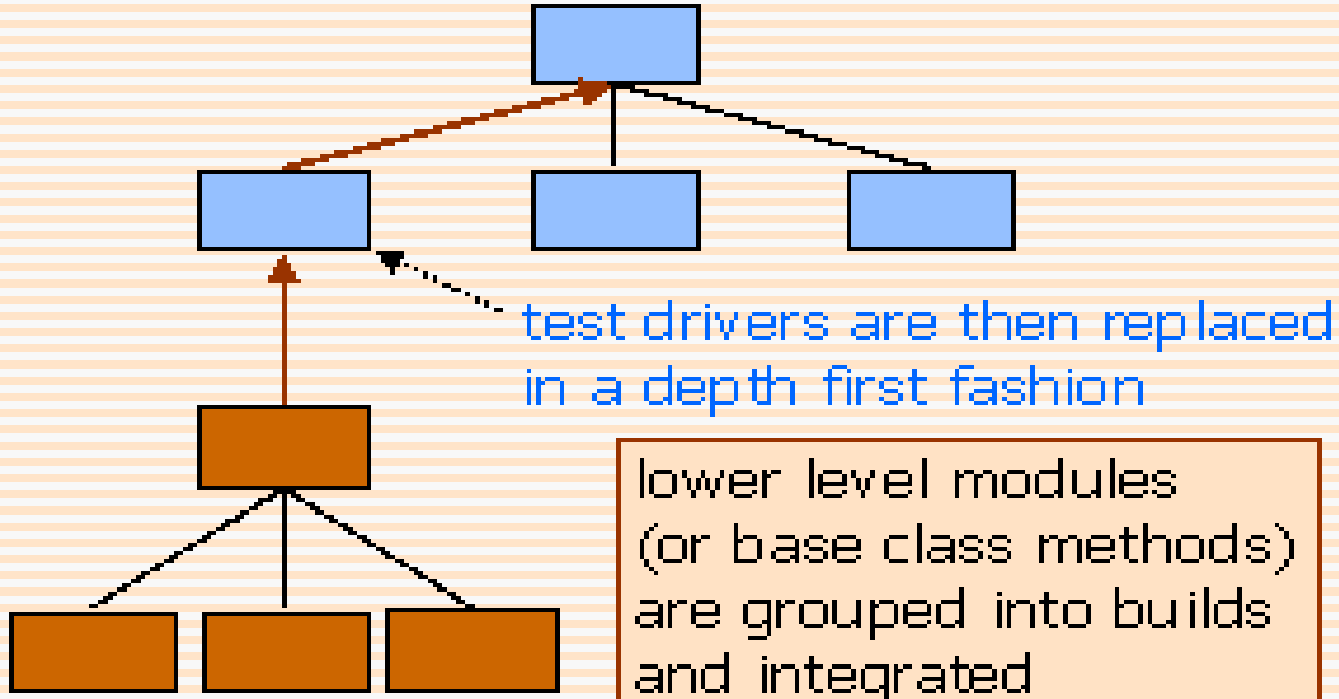
Stubs are replace one-at-a-time in "depth-first" fashion

As new modules (classes) are integrated, subsets of the tests will have to be rerun.

自底向上集成



Bottom-Up Integration



3.3.3 集成测试使用的测试方法

- 通常采用白盒测试与黑盒测试相结合的方法进行测试
- 在自底向上集成早期，白盒测试为主
- 随着集成规模越来越大，黑盒测试逐渐占据主导

3.4 集成测试原则

1. 所有公共接口都要被测试到
2. 关键单元必须进行充分的测试
3. 集成测试应当按一定的层次进行
4. 集成测试的策略选择应当综合考虑质量、成本和进度之间的关系
5. 集成测试应尽早开始，并以总体设计为基础
6. 在单元与接口的划分上，测试人员应该与设计人员进行沟通
7. 当接口发生修改时，涉及的相关接口必须进行再测试

提纲

- 软件测试级别概述
- 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）

4. 系统测试

- 系统测试概述
- 系统测试的准备工作
- 系统测试的任务

4.1 系统测试概述

- 系统测试是对集成测试通过后的软件，作为计算机系统的一个部分，与硬件、其它软件、数据、平台等结合起来，形成整个系统进行测试。
- 系统测试通常是消耗测试资源最多的地方
- 系统测试要测试系统的完整性及有效性

4.1.1 系统测试的内容

- 系统测试——软件系统是指交付给用户的完整软件产品，它可能由软件、硬件、用户手册、培训材料等组成。系统测试关注在最高集成条件下产生的软件缺陷
- 典型的系统测试包括很多类型的测试，例如：功能测试，易用性测试，安全性测试，可靠性和可用性测试，性能测试，备份和恢复测试，便携式测试以及其他测试等

4.2 系统测试的准备工作

- 在系统测试之前，要进行各种准备：

1. 收集需求说明书、参考手册等
2. 做好测试计划
3. 设计好测试案例
4. 组织好测试过程
5. 处理好测试结果

4.3 系统测试的任务

- 系统测试主要包括：

1. 功能测试（黑盒测试）
2. 用户界面测试
3. 性能测试
4. 可靠性和可用性测试
5. 恢复测试
6. 安全性测试（两种不同的安全：security和safety）
7. 强度测试
8. 易用性测试

4.3.1 性能测试

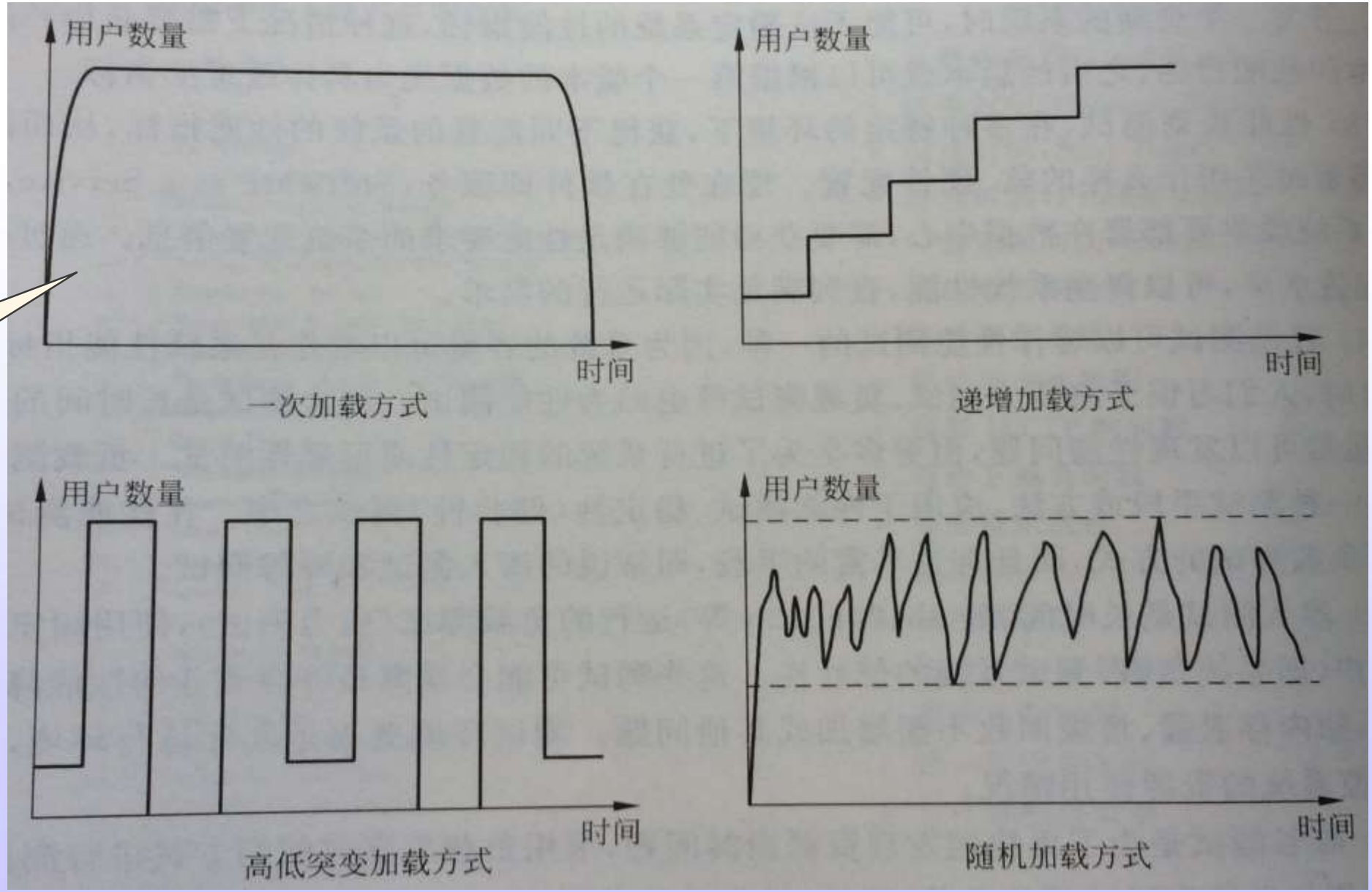
- 性能测试：Performance test，为了发现系统性能问题或获取系统性能相关指标（如：运行速度，响应时间，资源使用率等）而进行的测试。
- 一般在真实环境、特定负载条件下，通过工具模拟实际软件系统的运行及操作，同时监控性能各项指标，最后对测试结果进行分析，来确定系统的性能状况。

4.3.1.1 性能指标

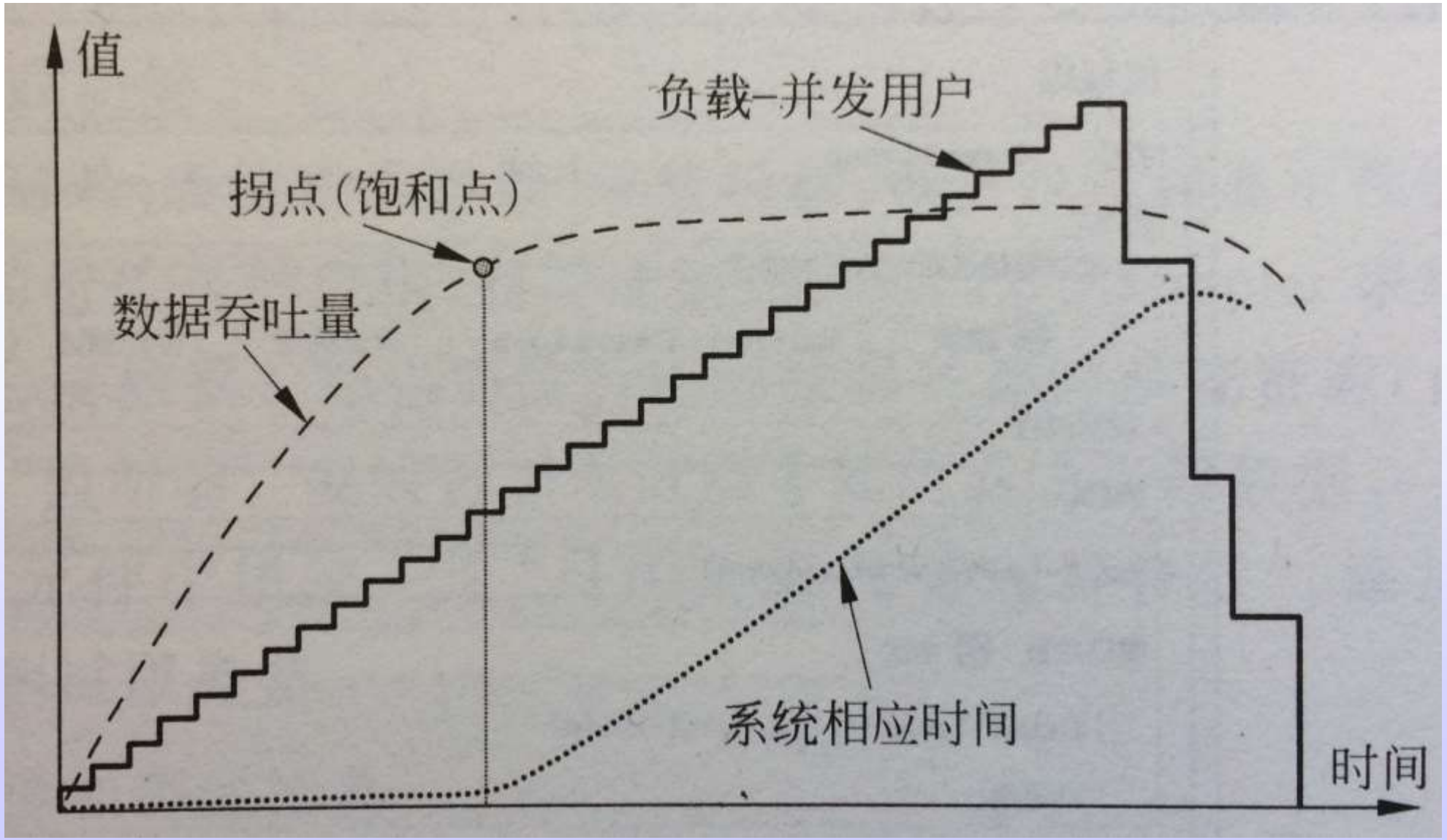
- 系统资源使用率
  - CPU、内存等资源使用率越低，系统性能越好。资源使用率是进行性能改善的主要依据
- 系统行为表现
  - 请求响应时间
  - 事务响应时间
  - 数据吞吐量
  - ……

4.3.1.2 系统负载模式

“Flat”测试：一次**加载所有的用户，**然后在预定的**时间段内持续**运行



4.3.1.3 性能测试结果分析



4.3.2 安全性测试

●安全性（**security**），系统不受攻击的安全性，检查系统对非法侵入的防范能力。测试人员假扮非法入侵者，采用各种办法试图突破防线。例如：

- 想方设法截取或破译口令；
- 专门开发软件来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，推导所需信息等等。

● 理论上讲，只要有足够的时间和资源，没有不可进入的系统。因此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。

4.3.2 安全性测试

●安全性（**safety**），软件对环境的安全性，关注的是避免软件对环境（主要是人）的危害，能否限制危害并恢复。

●软件安全是指避免导致软件环境中的操作人员受到损坏、伤害或丧失生命的状态，以及在进入恶劣状态时恢复和限制损坏的能力。

●软件已经开始控制生活中越来越多的设备，软件安全已经成为一个关键问题。

4.3.3 可靠性测试

- 可靠性（Reliability）是产品在**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性。
- 软件可靠性是软件系统的固有特性之一，它表明了一个软件系统按照用户的要求和设计的目标，执行其功能的可靠程度。
- 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。

提纲

- 软件测试级别概述
- 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）

5. 验收测试

- 验收测试概述
- 验收测试的主要内容
- α测试
- β测试

5.1 验收测试概述

- 也叫确认测试
- 验收测试在软件开发结束后，在产品发布之前进行
- 验收测试面向客户，从客户使用和业务场景的角度出发，目的是得到用户的理解和认同

5.2 验收测试的主要内容

- 验收测试主要完成以下内容：

1. 配置复审
2. 易用性测试
3. 安装与卸载测试
4. 可恢复性测试
5. 兼容性测试
6. 用户界面测试
7. 软件文档（用户手册、操作手册）检查
8. 软件功能和性能测试与测试结果的评审

用户界面测试

用户界面的7个要素:

安装与卸载测试

- 系统软件、应用软件、服务器、客户端等软件安装
- 产品升级安装
- 安装测试：环境要求清晰、提示信息合理、操作简单
- 卸载测试：卸载后资源释放、用户数据保留

可恢复性测试

- 当系统出错时，能否在指定时间间隔内修正错误或重新启动系统
- 可恢复测试首先要通过各种手段，让软件强制性地发生故障，然后验证系统是否能尽快恢复。
  - 对于自动恢复需验证重新初始化、检查点、数据恢复和重新启动等机制的正确性；
  - 对于人工干预的恢复系统，还需估测平均修复时间，确定其是否在可接受的范围内

5.3 α测试

- α测试——是由用户在开发环境下进行的测试，也可以是公司内部的用户在模拟实际操作环境下进行的测试
- α测试是在受控制的环境下进行的测试
- α测试的目的是评价软件产品的FURPS（Function、Usability、Reliability、Performance、Security，即功能、易用性、可靠性、性能和安全性）。

5.3.1 α测试的特点

1. 是在开发环境下进行的（环境是可控的）
2. 不需要测试用例评价软件使用质量
3. 用户往往没有相关经验，可以由开发人员或测试人员协助用户进行测试

- α测试的关键在于尽可能逼真地模拟实际运行环境和操作，并尽最大努力涵盖所有可能的用户操作方式。

5.4 β测试

- 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软件系统，并要求用户报告异常情况、提出批评意见，然后软件开发公司再对软件进行改错和完善
- β测试通常采用黑盒测试方法

5.4.1 β测试的特点

1. β测试是由软件用户在一个或多个用户的实际使用环境下进行的测试
2. β测试时开发人员通常不在测试现场
3. β测试是免费的
4. β测试的用户是任意的，环境是无法控制的

提纲

- 软件测试级别概述
- 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）

6 回归测试

- 回归测试概述
- 回归测试的策略

6.1 回归测试概述

- 回归测试是指软件被修改或扩充后重新进行软件测试
- 回归测试的目的是在程序有修改的情况下，保证原有功能正常的一种测试
- 回归测试不是一个测试阶段，而是一种可以用于单元测试、集成测试、系统测试及验收测试各个测试过程的测试技术
- 通常而言，程序的每次变动都将引起回归测试

6.2 回归测试的策略

- 回归测试时不一定要进行全面测试
  - 有选择的执行原来的部分测试
  - 补充新的测试用例，对修改或增加的功能进行测试
- 对已有的测试用例，回归测试可以采用不同的策略
  - 完全重复测试
  - 选择性重复测试

6.2.1完全重复测试

- 完全重复测试把所有的测试用例全部重新执行一遍，以确认缺陷修改的正确性和修改后周边是否受到影响
- 优点：是一种比较安全的策略，再测试全部用例具有最低的遗漏错误的风险
- 缺点：测试成本很高

6.2.2选择性重复测试

不同的选择策略：

1. 基于风险选择测试

- 基于一定的风险标准，从测试用例库中选择回归测试集。哪些部分被修改部分影响的可能性越大，越优先选择。

2. 基于操作剖面选择测试

- 优先选择那些针对最重要或最频繁使用功能的测试用例，释放和缓解最高级别的风险，有助于发现那些对可靠性有最大影响的故障

3. 再测试修改的部分

- 将回归测试局限于被改变的模块和它的接口上，尽可能覆盖受到影响的部分。这种策略效率高、风险大，需要良好的经验和深入的代码分析。

6.3 回归测试的基本过程

1. 识别软件中被修改的部分
2. 从原测试用例库T中，选择那些对新版本依然有效的测试用例，其结果是建立一个新的测试用例库T0
3. 依据策略从T0中选择测试用例进行测试
4. 如果必要，生成新的测试用例集（或修改已有测试用例集）T1，用于测试T0无法充分测试的软件部分
5. 用T1进行测试

- 第2、3步是验证修改是否破坏了现有的功能，第4、5步是验证修改工作本身

内容回顾

- 软件测试级别概述
- 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）

The End

Any Question?