

# 软件测试技术

# 6

## 集成测试和系统测试

- ❖ 6.1 集成测试前的准备
- ❖ 6.2 集成测试的模式
- ❖ 6.3 回归测试
- ❖ 6.4 功能测试
- ❖ 6.5 非功能测试

# 集成测试和系统测试

- 所有功能独立的模块经过单元测试后，接下来进行集成测试
  - ▣ 将单元模块按照设计要求组合起来进行测试
  - ▣ 检查单元间接口是否存在问题
- 由若干个不同测试组成，目的是充分运行系统，验证这个系统是否满足非功能性的质量需求
  - ▣ 是否能正常工作并完成相关任务
  - ▣ 是否能承受大批用户使用
  - ▣ 系统出错时，是否能很快恢复并转移故障
  - ▣ 是否能长期稳定的运行下去

# 6.1 集成测试准备时需要考虑的因素

## □ 人员安排

- ▣ 既需要熟悉单元的内部细节，又要求能够从足够高的层次上观察整个系统
- ▣ 由经验丰富的测试人员和软件开发人员协同完成

## □ 测试计划

- ▣ 系统设计阶段开始制定，在系统实施集成之前完成
- ▣ 包括的内容：测试的描述和范围、测试的预算、测试环境、集成次序、测试用例设计思想和时间表等

## □ 测试内容

- 重点测试内容包括个单元的接口是否吻合、代码是否符合规定的标准、界面标准是否统一等

## □ 集成模式

- 要么按照设计要求一次全部组装
- 要么以增量方式逐渐组装

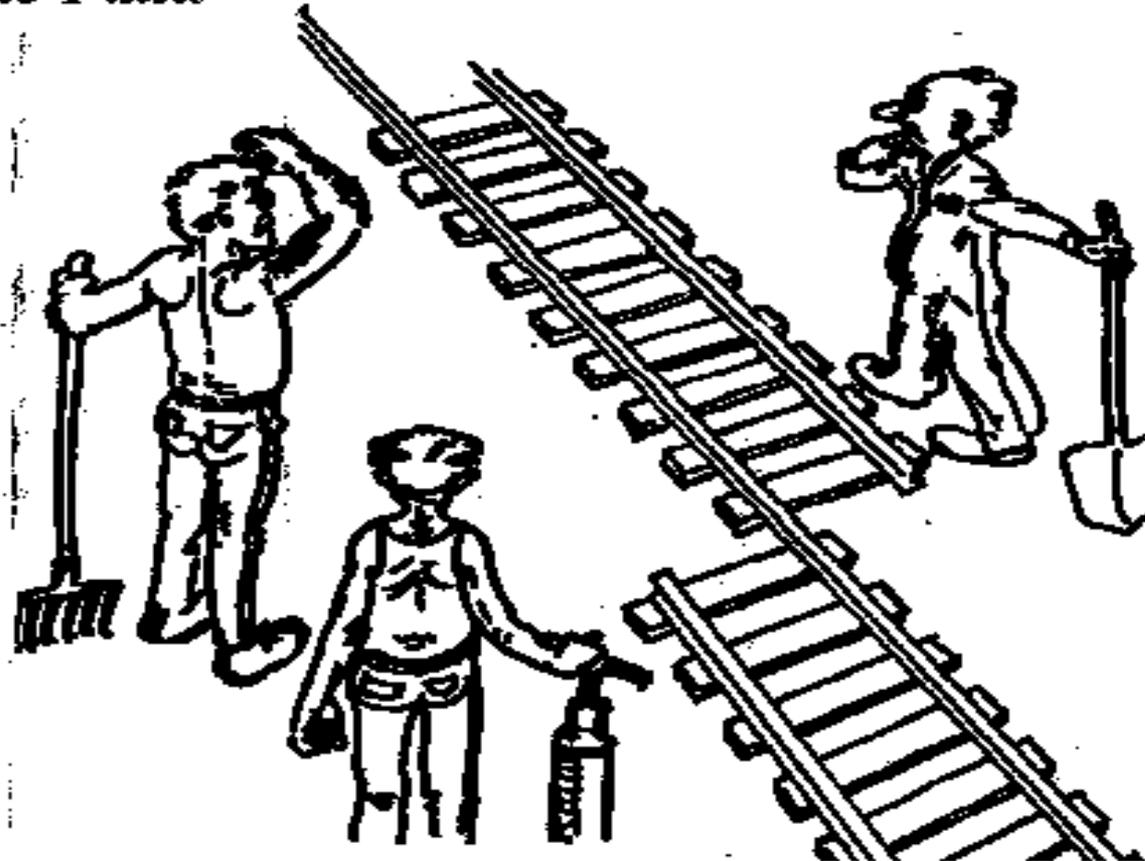
## □ 测试方法

- 以黑盒测试为主

# 为什么总是集成不起来？

6

## *Algorithmic Fault*



## 6.2 集成测试的模式

### □ 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

# 非渐增式测试模式Vs渐增式测试模式

- 渐增式测试模式需要编写的软件较多、工作量较大，而非渐增式测试开销小
- 渐增式测试模式发现模块间接口错误早，而非渐增式测试模式晚
- 非渐增式测试模式发现错误，较难诊断，而渐增式测试模式，如果发现错误则往往和最近加入的模块相关
- 渐增式测试模式测试更彻底
- 渐增式测试模式需要较多的机器时间
- 使用非渐增式测试，可以并行测试

# 常见的集成方法

9

- 自顶向下
- 自底向上
- 混合策略
- 大棒集成方法
- 三明治集成方法

# 自顶向下集成方法

- 从主控模块开始，沿着软件的控制层次向下移动，从而逐渐把各个模块结合起来。
- 具体步骤
  - ① 对主控模块进行测试，测试时用桩程序代替所有直接附属于主控模块的模块
  - ② 根据选定的结合策略（深度优先或广度优先），每次用一个实际模块代替一个桩程序
  - ③ 在结合下一个模块的同时进行测试
  - ④ 为了保证加入模块没有引进新的错误，可能需要进行回归测试
  - ⑤ 重复步骤2，直至完成

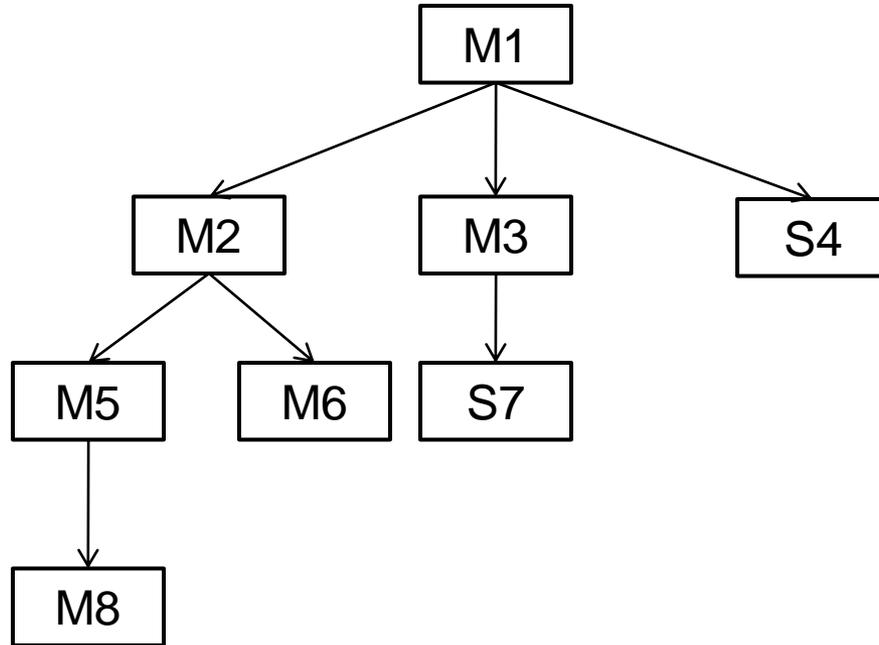
# 常见的两种策略

11

- 策略一：深度优先策略
- 策略二：广度优先策略

# 深度优先遍历策略

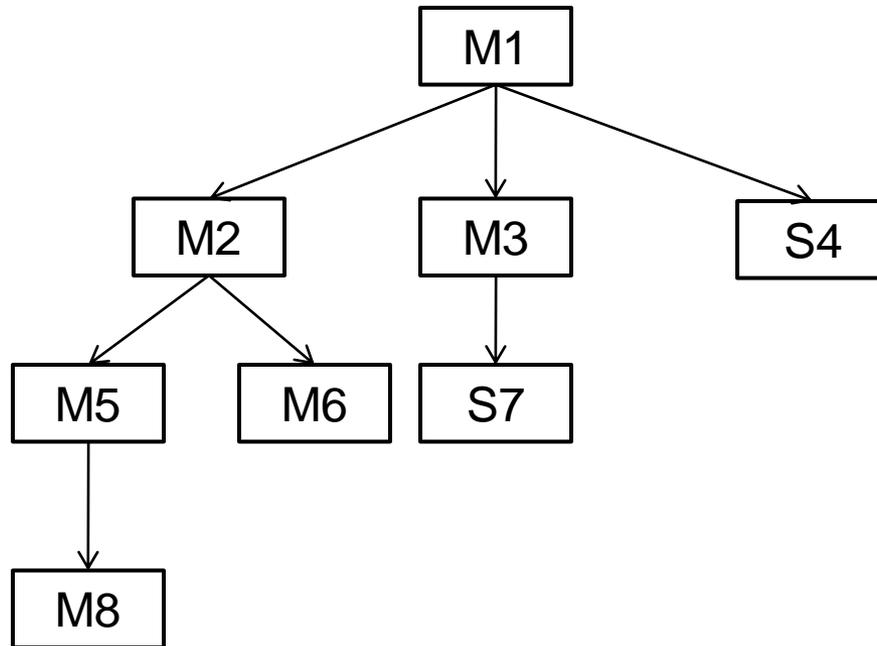
12



M1->M2->M5->M8->M6->M3->S7->S4

# 广度优先遍历策略

13



M1->M2->M3->S4->M5->M6->S7->M8

# 自顶向下集成方法的优点和不足

14

## □ 优点

- 不需要测试驱动程序
- 在测试阶段早期实现并验证系统的主要功能，而且能在早期发现上层模块的接口错误

## □ 不足

- 需要桩程序
- 底层关键模块的错误发现较晚
- 在早期不能充分展开人力

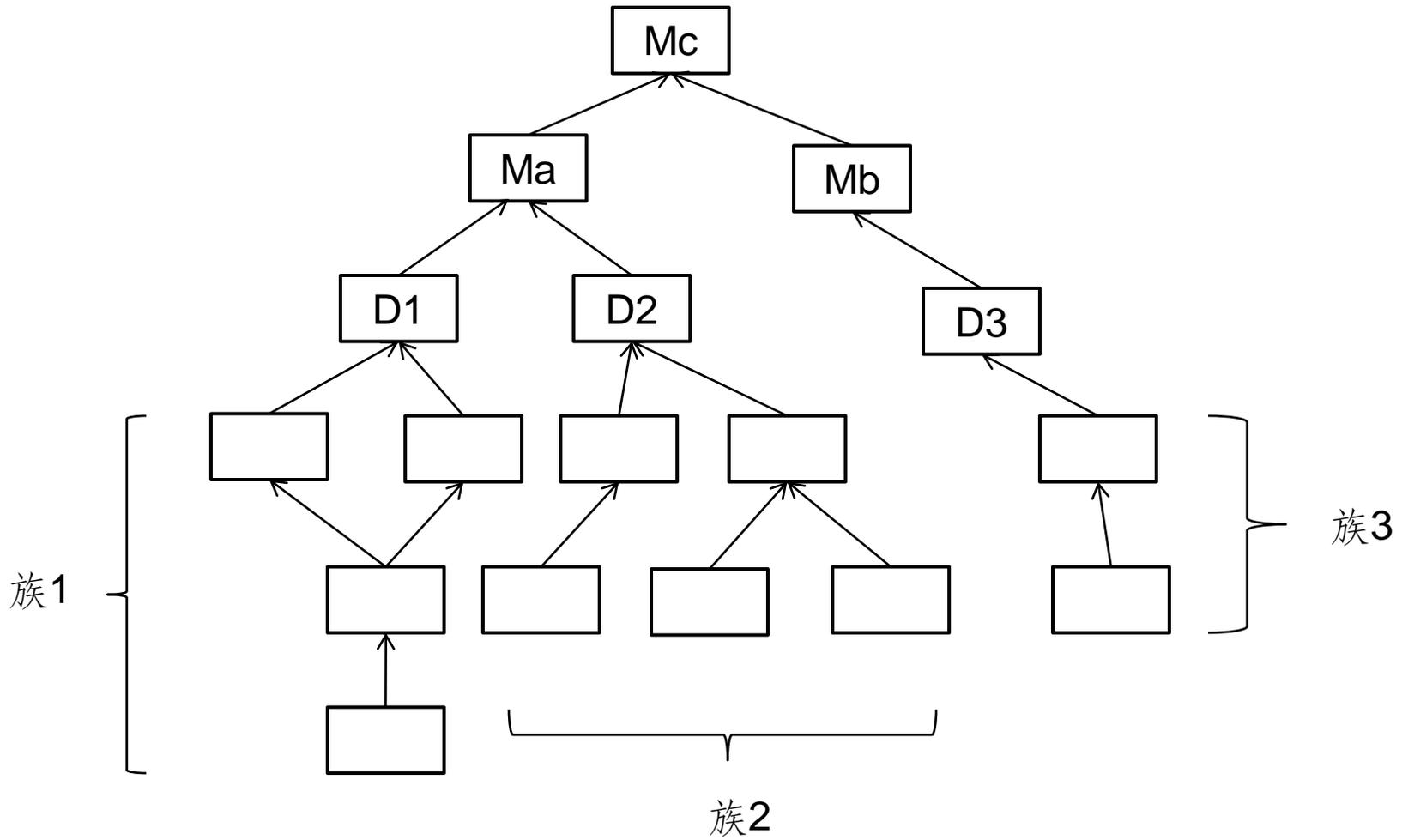
# 自底向上法

15

- 从软件结构最底层的模块开始集成以进行测试
- 步骤
  - ① 把底层模块组合成实现某个特定的软件子功能的族
  - ② 写一个驱动程序，协调测试数据的输入和输出
  - ③ 对由模块组成的子功能族进行测试
  - ④ 去掉驱动程序，沿软件结构自下向上移动，把子功能组合起来形成更大的子功能族
  - ⑤ 重复步骤2，直至完成
- 优点和不足与自顶向下方法相反

# 实例

16



# 混合策略

17

## □ 改进的自顶向下法

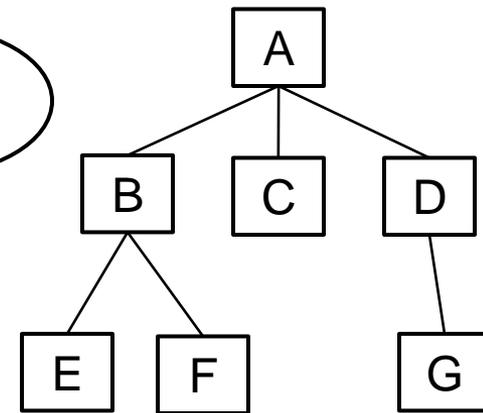
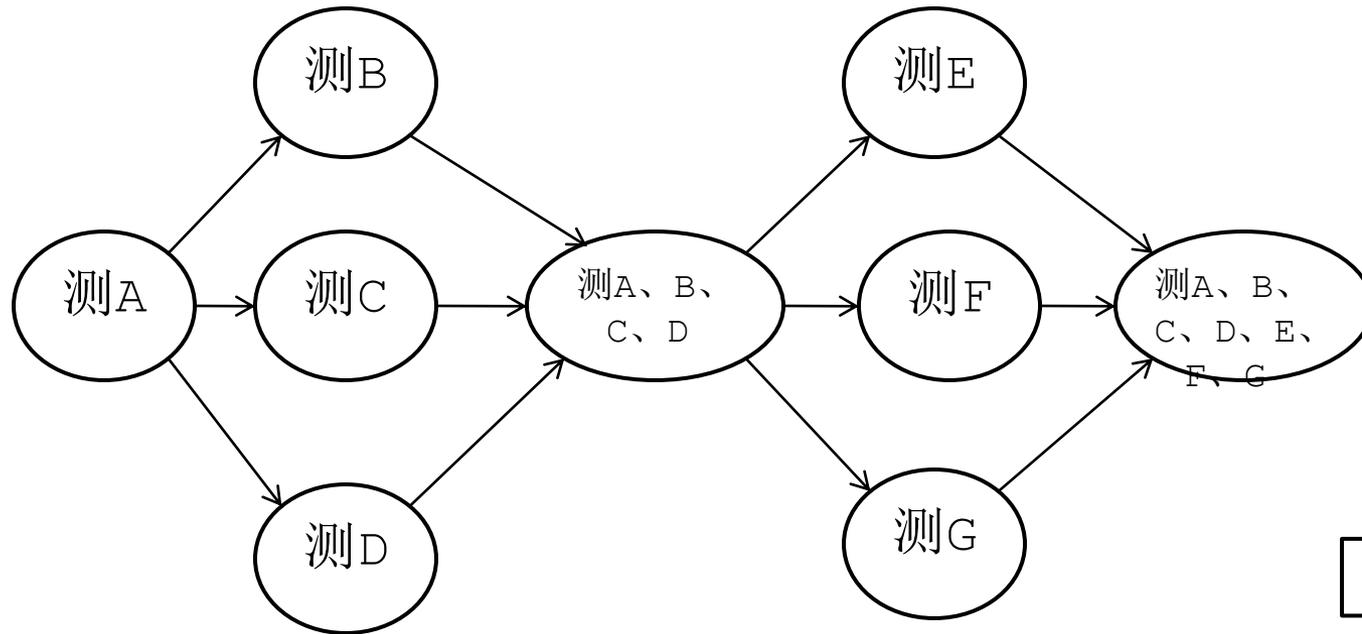
- 基本使用自顶向下法，但在测试早期，使用自底向上法测试少数的关键模块

## □ 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，两者结合

# 实例

18



# 大棒集成方法 (Big-bang Integration)

19

## □ 方法描述

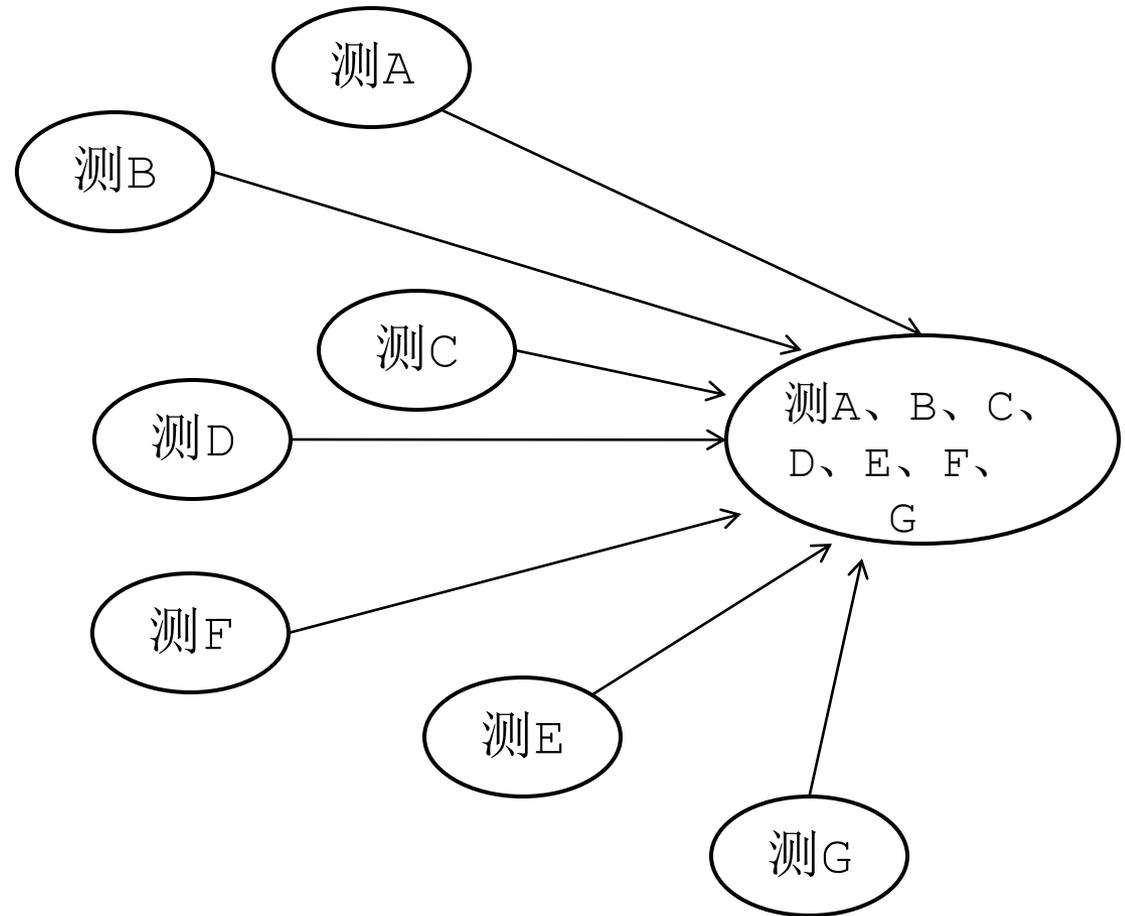
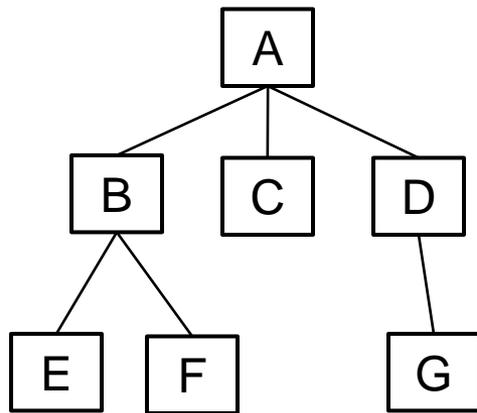
- ▣ 先对每一个子模块进行测试（单元测试），然后将所有子模块一次性的全部集成起来进行集成测试。

## □ 不足

- ▣ 所有模块一次集成，所以很难确定错误的真正位置、所在的模块和错误的原因

# 实例

20



# 三明治集成方法 (Sandwich Integration)

21

## □ 方法描述

- ▣ 自两头向中间集成

## □ 优点

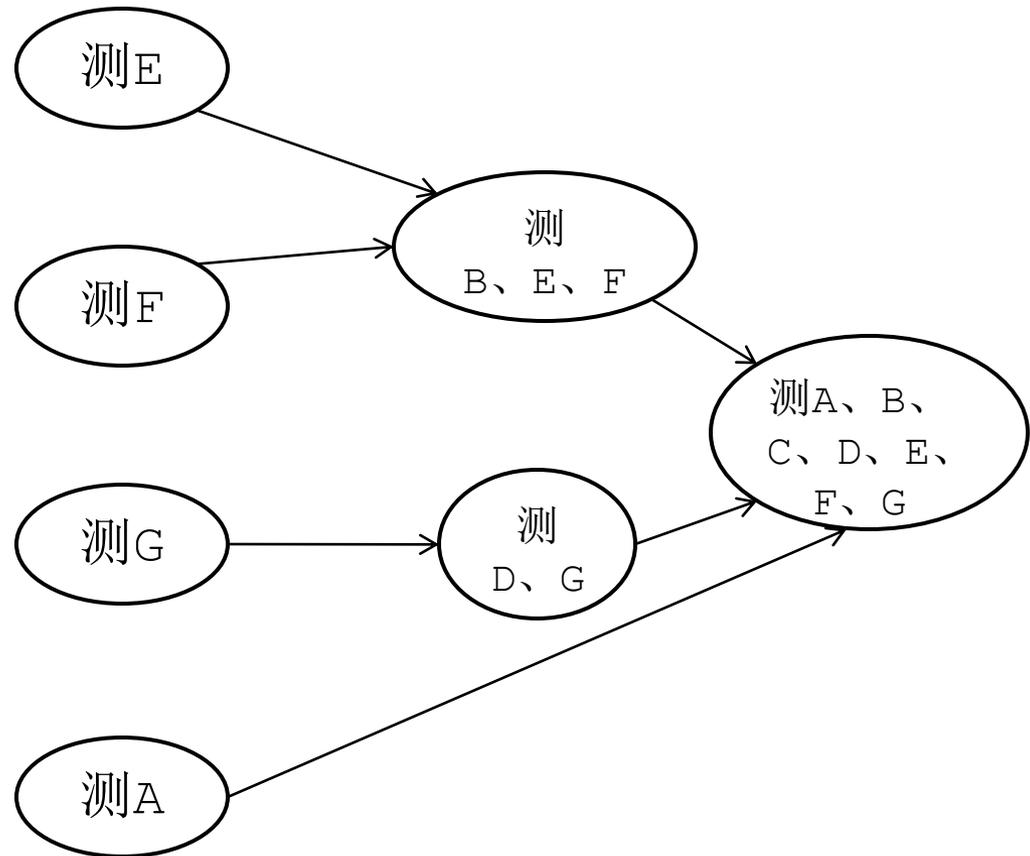
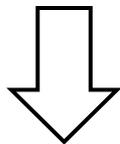
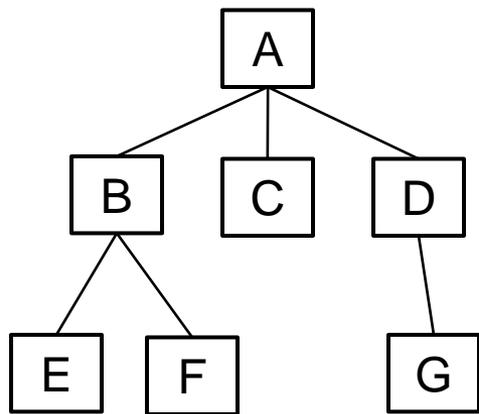
- ▣ 将自顶向下和自底向上的集成方法有机结合起来，不需要写桩程序，因为在测试初自底向上集成已经验证了底层模块的正确性

## □ 缺点

- ▣ 在真正集成之前每一个独立的模块没有完全测试过

# 实例

22



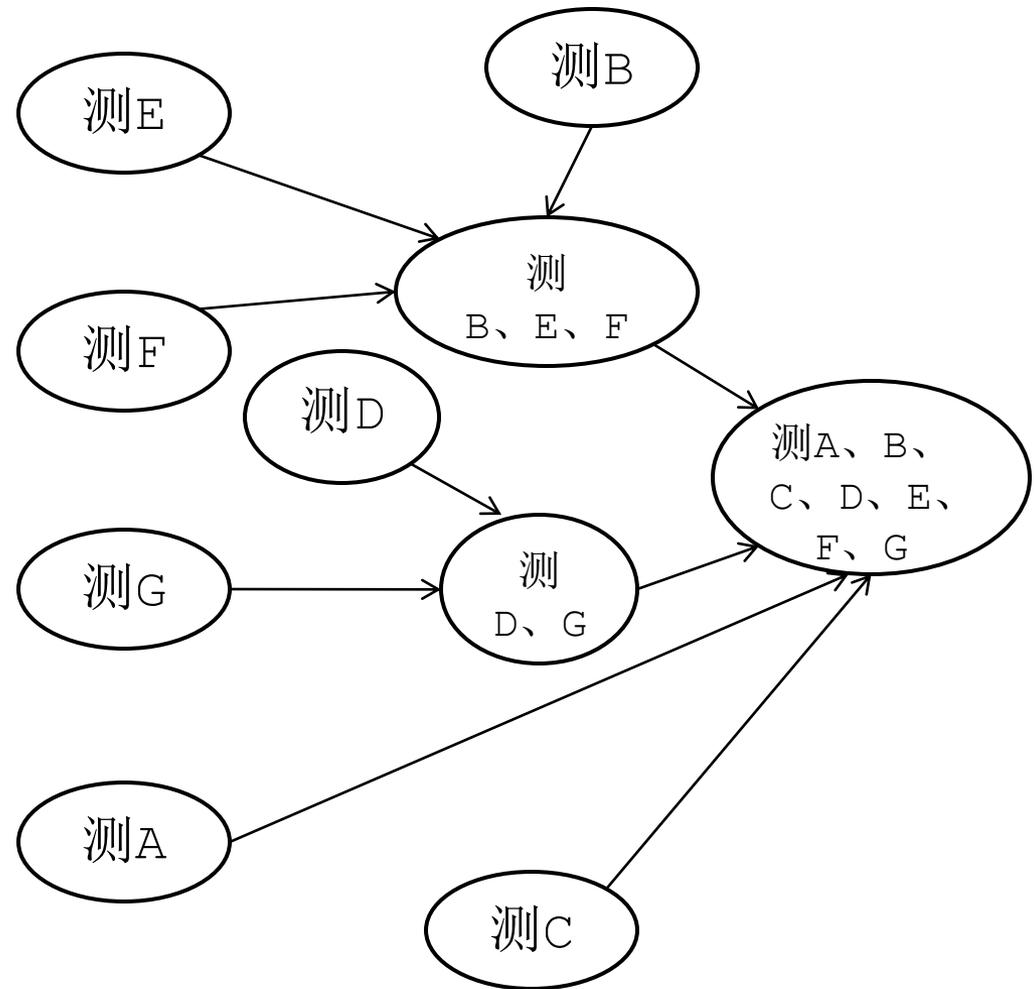
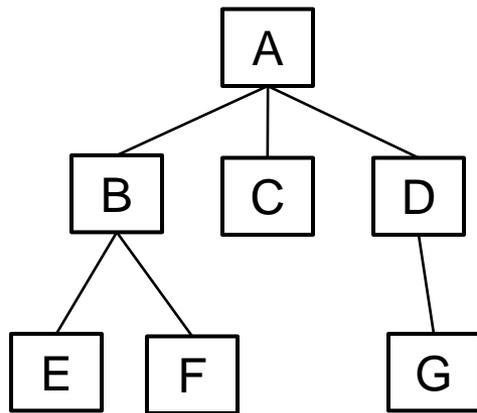
# 改进的三明治集成方法

23

- 不仅自两头向中间集成，而且保证每个模块均得到单独的测试，使得集成测试进行的比较彻底

# 实例

24



# 综合比较

25

	自底向上	自顶向下	混合策略	大棒	三明治	改进的三明治
集成	早	早	早	晚	早	早
基本程序能工作时间	晚	早	早	晚	早	早
需要驱动程序	是	否	是	是	是	是
需要桩程序	否	是	是	是	是	是
工作并行性	中	低	中	高	中	高
特殊路径测试	容易	难	容易	容易	中等	容易
计划与控制	容易	难	难	容易	难	难

# 持续集成

- 通常系统集成都会采用持续集成的策略，软件开发中各个模块不是同时完成，根据进度将完成的模块尽可能早的进行集成，有助于尽早发现缺陷，避免集成中大量缺陷涌现
- 采用持续集成方式，有助于缺陷及早发现和定位，提高软件开发的质量和效率

# 6.3 回归测试

- 适用测试场景
  - ▣ 新增用户需求或用户需求发生变更
  - ▣ 修正缺陷
  - ▣ 性能提升需要
- 回归测试的目的
  - ▣ 保障修改的正确性
  - ▣ 避免修改对被测程序其他模块产生副作用
- 回归测试在软件开发过程中占据很大比重，在软件开发各个阶段都需要进行回归测试。在极限编程方法中，更是要求每天都进行若干次回归测试

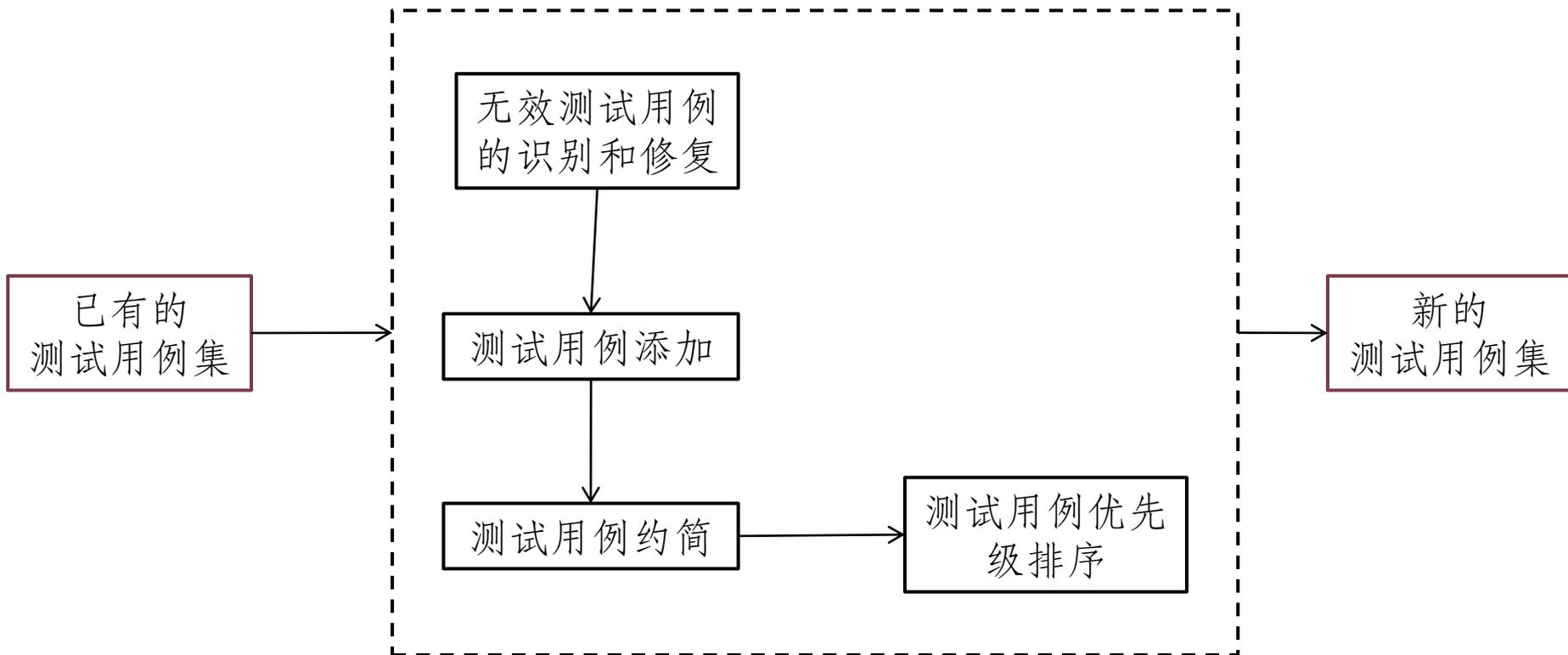
# 回归测试的基本流程

28

- ❑ 一个简单策略：重新执行已有测试用例集
  - ❑ 但该策略并不可行
- ❑ 目前采用的策略
  - ① 识别出软件中的修改部分
  - ② 从已有的测试用例集中排除不再适用的测试用例。例如：测试用例的输入发生改变、测试用例的预期输出发生改变；尝试修复部分测试用例
  - ③ 从已有的测试用例集中选择出与修改相关的测试用例
  - ④ 添加新的测试用例满足指定的覆盖准则
  - ⑤ 依据测试用例的检错能力进行排序
  - ⑥ 执行新的测试用例集

# 回归测试的简易流程图

29



# 6.4 功能测试

30

- 在单元测试阶段
  - ▣ 保证测试的独立模块功能正确
- 在集成和系统测试阶段
  - ▣ 不仅要考虑模块间的相互作用，而且要考虑系统的应用环境

# 6.5 非功能测试

- 与功能测试不同，主要测试软件的非功能属性，包括：
  - 性能测试
  - 压力测试
  - 容量测试
  - 安全性测试
  - 可靠性测试
  - 容错性测试

# 性能测试 (Performance Test)

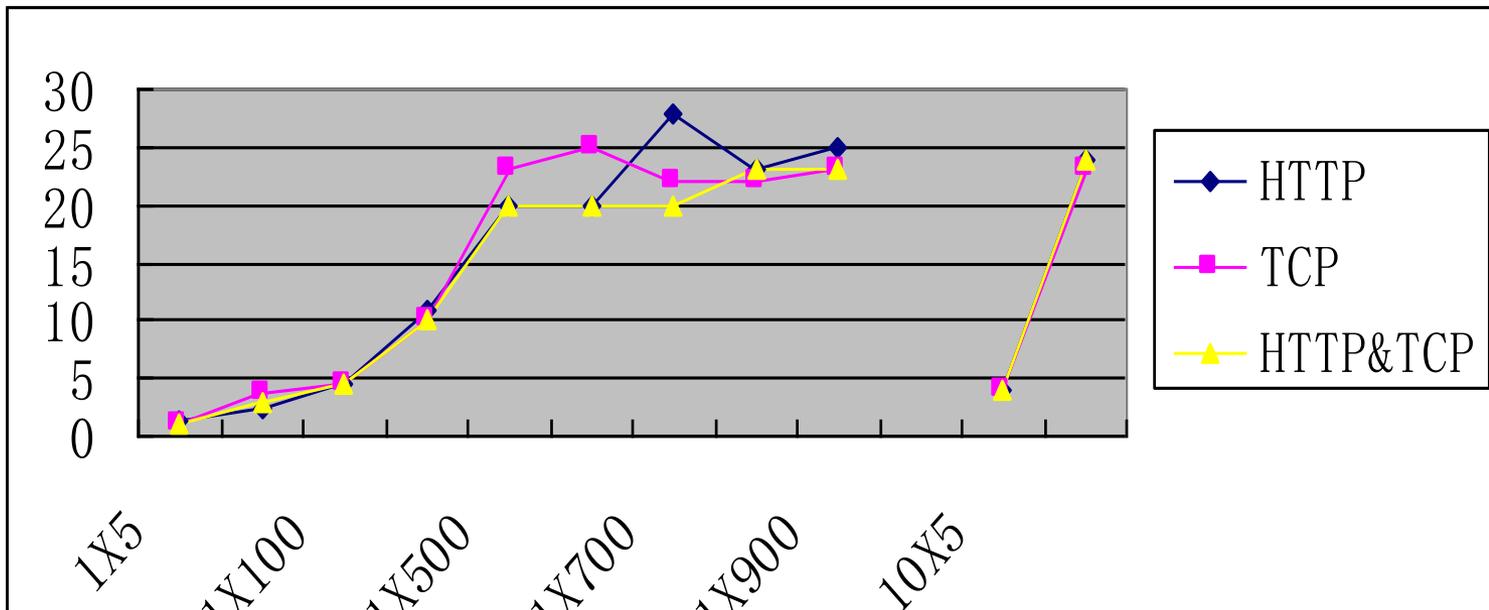
32

## □ 针对实时系统和嵌入式系统

▣ 不仅需要满足功能要求，而且还需要满足性能要求

## □ 性能测试

▣ 通过测试以确定系统运行时的性能表现，如得到运行速度、响应时间、系统资源占用情况等方面的系统数据



1\*5表示：  
1个课程，  
五个在线  
用户

# 系统负载的概念 (1)

33

- 在线用户：通过浏览器访问登陆Web应用系统后并且没有退出该系统的用户
  - ▣ 通常对应Web应用服务器的一个Session
- 虚拟用户：模拟浏览器向Web服务器发送请求并接收响应的一个进程或线程
- 并发用户
  - ▣ 严格意思上：这些用户同一时刻应该做同一事情
  - ▣ 不严格的说：同一时刻可做不同的事情
  - ▣ 性能测试中往往采用严格意义上的说法



# 系统负载的概念 (2)

34

- 用户并发数量：上述并发用户的数量
- 思考时间：浏览器在收到响应后到提交下一个请求间的间隔时间
  - ▣ 思考时间越短，服务器承受的负载越重
- 负载模式：既加载方式
  - ▣ 例如：一次建立200个并发连接，还是每秒逐渐增加十个连接，直至增加到200个。
  - ▣ 其他加载方式：随机加载、峰谷交替加载



# 系统性能指标

35

- 包括系统资源使用率和系统行为表现
  - ▣ 资源使用率越高、系统性能表现越差
  - ▣ 资源使用率越低、系统性能表现越好
- 包含的指标
  - ▣ **请求响应时间**：浏览器向Web服务器提交一个请求到收到响应之间的间隔时间。
    - 在测试用具中又称为TLB（Time to Last Byte）
  - ▣ **事务响应时间**：是这些请求完成处理所花费的时间。
    - 事务可能由一系列请求组成
  - ▣ **数据吞吐量**：单位时间内客户端和服务端之间网络上传输的数据量。

# 性能测试的基本过程

36

- ① 确定性能测试需求：确定性能指标和关键操作
- ② 根据测试需求，选择测试工具和开发相应的测试脚本
- ③ 建立性能测试负载模型：确定并发虚拟用户的数量、每次请求的数据量、思考时间、加载方式和持续加载时间等；通常不会一次设计到位，而是一个不断迭加过程
- ④ 执行性能测试：多次运行性能测试负载模型，获得系统的性能数据，发现性能瓶颈
- ⑤ 提交性能测试报告：包括性能测试方法、负载模型和实际执行的性能测试、性能测试结果极其分析

# 性能测试要点

- 测试环境应尽量与产品运行环境保持一致，应单独运行尽量避免与其他软件同时使用。
- 性能测试一般使用测试工具和测试人员编制测试脚本来完成。
- 性能测试的重点在于前期数据的设计与后期数据的分析。
- 性能测试的用例主要涉及到整个系统架构的问题，所以测试用例一旦生成，改动一般不大，所以做性能测试的重复使用率一般比较高。

# 压力测试 (Stress Test)

38

- 又称为强度测试、负载测试
- 模拟实际应用的软硬件环境及用户使用过程的系统负荷，长时间或超大负荷的运行被测软件，来测试系统的性能、可靠性和稳定性等
- 目的
  - 通过执行可重复的负载测试，了解系统可靠性、性能瓶颈等。避免系统宕机带来的损失

# 压力测试

39

## □ 针对异常情况的设计

- 异常情况包括峰值、大量数据的处理能力、长时间运行等情况

## □ 例如

- 当中断的正常频率为每秒1或2个时，运行每秒产生10个中断的测试用例
- 定量的增长数据输入频率，检查对数据处理的反应能力
- 运行需要最大存储空间（或其他资源）的测试用例
- 运行可能导致虚拟操作系统崩溃或大量数据对磁盘进行存取操作的测试用例

# 步骤 (1)

## □ 测试压力估算

- 根据产品说明书的设计要求或以往版本的实际运行经验对测试压力进行估算，给出合理的估算结果
- 例如：
  - 如果单台服务器实际使用时一半只有100个并发用户，但在某一时间段用户峰值可达到500个
  - 则事先预测的压力值为500个用户的1.5~2倍

# 步骤 (2)

## □ 测试环境准备

- 包括硬件环境（服务器、客户机等）、网络环境（网络通信协议、带宽等）、测试程序（能正确模拟客户端的操作）和数据准备
- 分析压力测试中系统容易出现瓶颈的地方，从而有目的的调整测试策略或测试环境，使压力测试结果真实的反映出软件的性能
- 包括：压力稳定性测试和破坏性加压测试
  - **压力稳定性测试**：在选定的压力值下，持续运行24小时以上进行稳定性测试
  - **破坏性加压测试**：在压力稳定性测试中可能会出现一些问题，如系统性能明显降低，但仅从以上测试很难暴露出真实的原因，通过破坏性不断加压的手段，往往能快速造成系统的崩溃或让问题明显的暴露出来

# 步骤 (3)

## □ 问题的分析

- 压力测试往往采用黑盒测试方法，测试人员很难对出现的问题进行准确定位，开发人员又没有相应的环境和时间去重现问题，所以需要适当的分析和详细的记录
- 查看服务器上的进程及相应的日志文件可能立刻找到问题的关键
- 查看监视系统性能的日志文件，找出文件出现的关键时间
- 检查测试运行参数，进行适当调整重新测试，看看是否能够再现问题
- 对问题进行分解，屏蔽某些因素或功能，试着重现问题

# 步骤 (4)

## □ 累积效应

- 如果测试人员在压力测试中重启系统，这是一个不好的习惯，会忽略累积效应，使得一些缺陷无法被发现
- 例如某进程每次调用时申请占用的内存在运行完毕时并没有完全释放

# 容量测试 (Capacity Test)

44

- 预先分析出反映软件系统应用特征的某项指标的极限值，例如：
  - ▣ Web站点可以支持的并发用户的访问量
  - ▣ 网络在线会议系统的与会者人数
- 知道了系统的实际容量
  - ▣ 如果不满足要求，应该去寻找新的解决方案
  - ▣ 若没有新的解决方案，必须在产品发布说明书上明确容量限制，避免产品使用纠纷

# 容量测试

- 确定软件系统还能保持主要功能正常运行的某项指标的极限值
  - ▣ 如最大并发用户数量、数据库记录数等
- 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
  - ▣ 数据库包含的记录数目

# 容量测试的完成标准

46

- 所执行的测试已全部执行，而且达到或超出指定系统限制时没有出现任何软件故障
  - ▣ 注意：选择不同的加载策略可以反映不同状况下的容量，例如针对网上聊天室软件
    - 一个聊天室内1000个用户
    - 100个聊天室，每个聊天室内10个用户

# 安全性测试

47

- ISO 8402对安全性（Safety）的定义
  - ▣ 使伤害或损害的风险限制在可接受的水平内
- 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，采用各种方法试图突破防线，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

- 理论来讲，只要有足够的时间和资源，没有不可进入的系统
- 系统安全设计的准则是
  - 是非法入侵的代价超过被保护信息的价值，此时非法入侵者以无利可图

# 两种级别的安全性

- 包括应用程序级别的安全性和系统级别的安全性
- 关系
  - 应用程序级别的安全性，包括对数据或业务功能的访问；系统级别的安全性，包括对系统的登录和远程访问。
  - 应用程序级别的安全性可确保，在预期的安全性情况下，操作者只能访问特定的功能或用例，或者只能访问有限的的数据。例如：某财务系统，只有管理员可以删除用户或数据
  - 系统级别的安全性可确保只有具备系统访问权限的用户才能访问应用程序，而且只能通过相应的网关来访问

# 测试目标

50

- 应用程序级别的安全性
  - 核实操作者只能访问其所属用户类型已被授权访问的那些功能或数据
- 系统级别的安全性
  - 核实只有具备系统和应用程序访问权限的操作者才能访问系统和应用程序

# 测试范围

51

- 确定并列出具各用户类型及其被授权访问的功能或数据
- 为各用户类型创建测试，并通过创建各用户类型所特有的事务来核实其权限
- 修改用户类型并为相同的用户重新运行测试
- 对于每种用户类型，确保正确的提供或拒绝这些附加的功能或数据

# 安全性测试方法

52

## □ 静态的代码安全测试

- 通过扫描源代码，根据程序中数据流、控制流等信息与其特有软件安全规则库进行匹配，从中找出代码中潜在的安全漏洞

## □ 动态的渗透测试

- 使用自动化工具或者人工的方法模拟黑客的输入，对应用系统进行攻击性测试，从中找出运行时刻所存在的安全漏洞

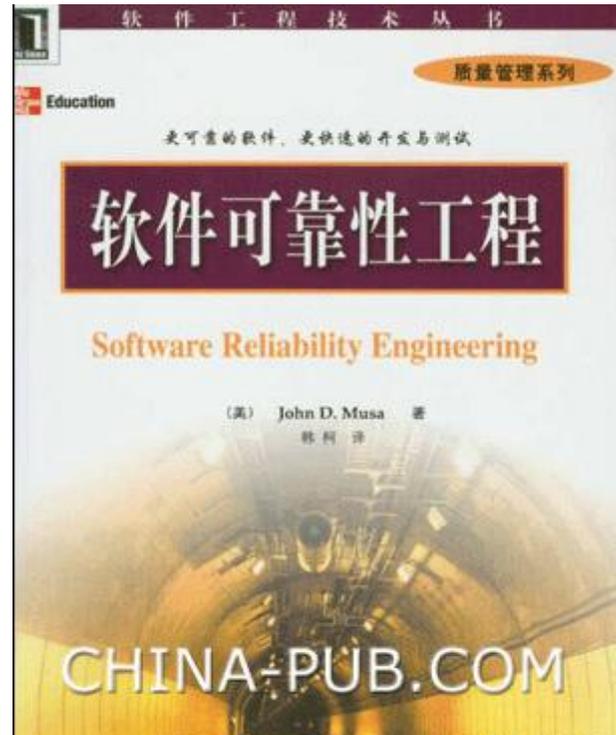
## □ 程序数据扫描

- 进行内存测试，内存测试可以发现诸如缓冲区溢出之类的漏洞

# 可靠性测试 (Reliability Test)

53

- 软件可靠性：是软件系统在规定的时间内及规定的环境条件下，完成规定功能的能力，表明一个软件系统按照用户要求和设计的目标，执行其功能的可靠程度



# 可靠性包含的三个要素

54

## □ 规定的时间

- 将“运行时间”作为“规定时间”的度量
- 运行时间包括软件系统运行后工作于挂起的累计时间

## □ 规定的环境条件

- 环境条件指软件的运行环境
- 不同环境下可靠性是不同的

## □ 规定的功能

- 可靠性与规定的**操作版本 (Operation Profile)** 有关
- 不同的操作版本调用的子模块并不一致，可靠性也不一样

# 可靠性模型

- 软件可靠性评估离不开软件可靠性模型，一般分为两类：
  - 软件可靠性结构模型
    - 依据系统结构逻辑关系，对系统的可靠性特征及其发展变化规律做出可靠性评价
    - 可用于软件可靠性综合评价和软件可靠性分解
  - 软件可靠性预计模型
    - 用来描述软件失效与软件缺陷的关系，可以对软件的可靠性特征做出定量的估计或评估
    - 依据软件缺陷与操作版本数据，利用统计学原理建立两者之间的数学关系

# 软件可靠性预计模型

56

## □ 面向时间的预计模型

- ▣ 以时间为基准，描述软件可靠性特征随时间变化的规律

## □ 面向输入数据的预计模型

- ▣ 描述软件可靠性与输入数据的联系，利用程序运行中的失效次数与成功次数的比作为软件可靠性的度量

## □ 面向错误数的预计模型

- ▣ 描述程序中现存错误数的多少，预示程序的可靠性

# 可靠性数据收集

57

- 用时间定义的软件可靠性数据可分为四类
  - 失效时间数据：记录发生一次失效所累积经历的时间
  - 失效间隔时间数据：记录每次失效与上一次失效之间的时间间隔
  - 分组数据：记录某个时间区间发生了多少次失效
  - 分组时间内的累积失效数：记录某个区间内的累积失效数

# 容错性测试 (Recovery Test)

58

- 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
- 容错性测试包括
  - ▣ 输入异常数据或进行异常操作，以检验系统的保护性
  - ▣ 灾难恢复性测试

# 故障转移与数据恢复

59

## □ 故障转移

- 确保测试对象在出现故障时能成功完成故障的转移，并能从导致意外数据损失或数据完整性破坏的各种硬件、软件和网络故障中恢复

## □ 数据恢复

- 可确保：对于必须持续运行的系统，一旦发生故障，备用系统将不失时机的顶替发生故障的系统，以避免丢失任何数据或事务

# 测试目标

60

- 确保恢复进程将数据库、应用程序和系统正确的恢复到预期的已知状态。
- 测试中包括以下各种情况
  - ▣ 客户机断电、服务器断电
  - ▣ 通过网络服务器产生的通信中断或控制器被中断
  - ▣ 断电或与控制器的通信中断周期未完成
  - ▣ 数据库指针或关键字无效，数据库中的数据元素无效或遭到破坏

# 作业

61

- 1、简要介绍常见的五种集成方法：
  - 自顶向下、自底向上、混合策略、大棒集成方法和三明治集成方法
- 2、术语介绍：
  - 性能测试、压力测试、容量测试、安全性测试、可靠性测试和容错性测试

谢谢