
Integration and System Testing

概念

1. Integration testing: putting the pieces together
2. System testing: function and non-function （实际测试经常是performance）

Integration Strategies

- Top-down
- Bottom-up
- 混合策略
- Critical-first关键现行
- Function-at-a-time
- Big bang

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

- ◇人员安排
- ◇测试计划
- ◇测试内容
- ◇集成模式
- ◇测试方法

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。

非渐增式测试模式

- *大棒集成方法*

大棒集成方法(Big-bang Integration)

因为所有的模块一次集成的，所以很难确定出错的真正位置、所在的模块、错误的原因。这种方法并不推荐在任何系统中使用，适合在规模较小的应用系统中使用。

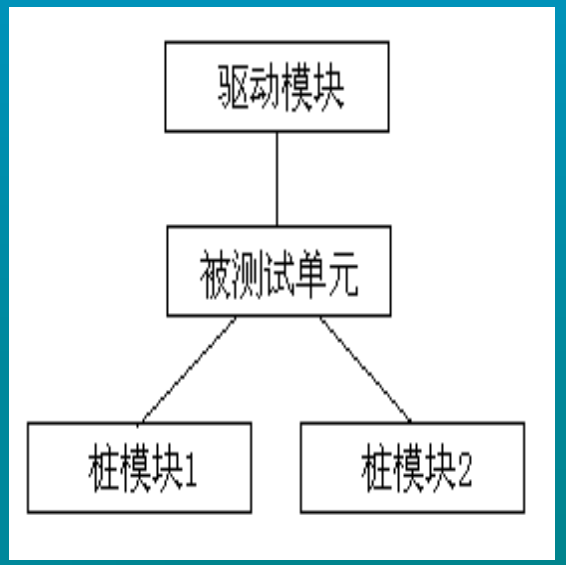
渐增式测试模式

- *自顶向下集成测试*
- *自底向上集成测试*
- *混合策略测试*
- Critical-first *测试*
- Function-at-a-time *测试*（对整个系统来讲是渐增式，但是对于单个功能来讲是非渐增式）

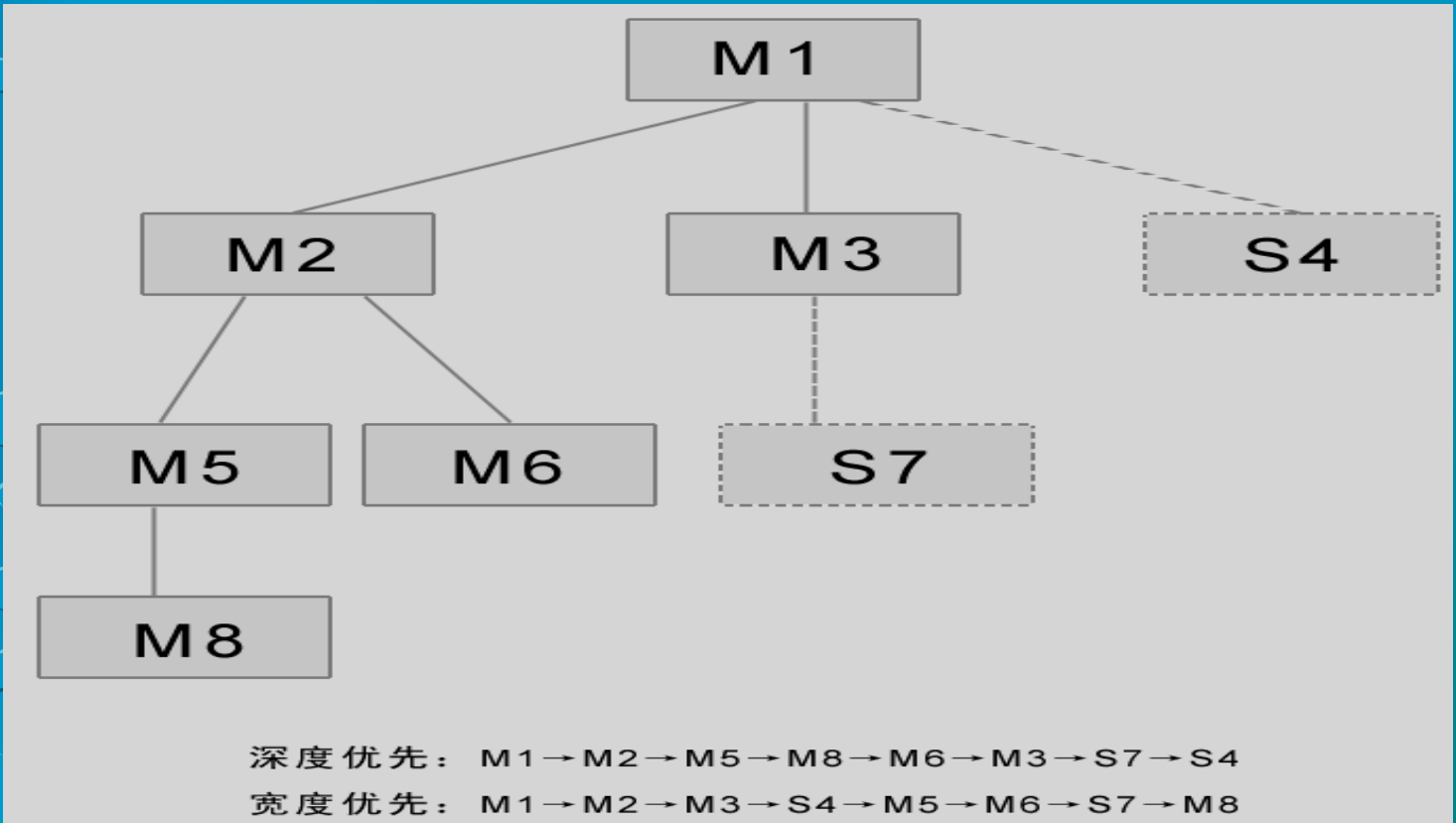
自顶向下和自底向上集成方法

驱动程序/驱动模块（driver），用以模拟被测模块的上级模块。驱动模块在集成测试中接受测试数据，把相关的数据传送给被测模块，启动被测模块，并打印出相应的结果。

桩程序/桩模块（stub），也有人称为存根程序，用以模拟被测模块工作过程中所调用的模块。桩模块由被测模块调用，它们一般只进行很少的数据处理，例如打印入口和返回，以便于检验被测模块与其下级模块的接口



自顶向下法(Top-down Integration)



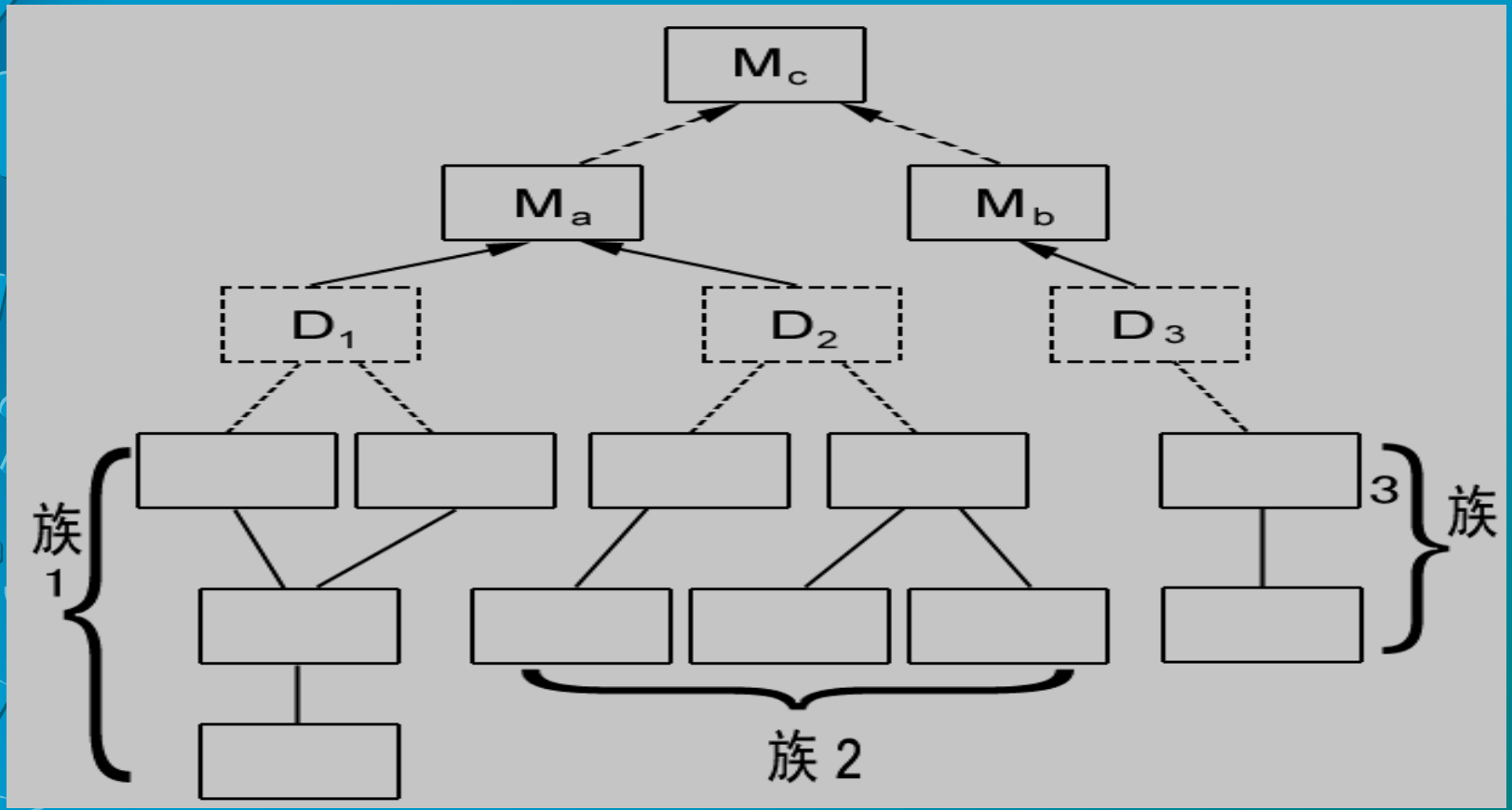
自顶向下法的主要优缺点：早期验证主要功能；

需要桩程序、上层模块接口错误发现早，底层关键模块错误发现晚

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

自底向上法(Bottom-up Integration)

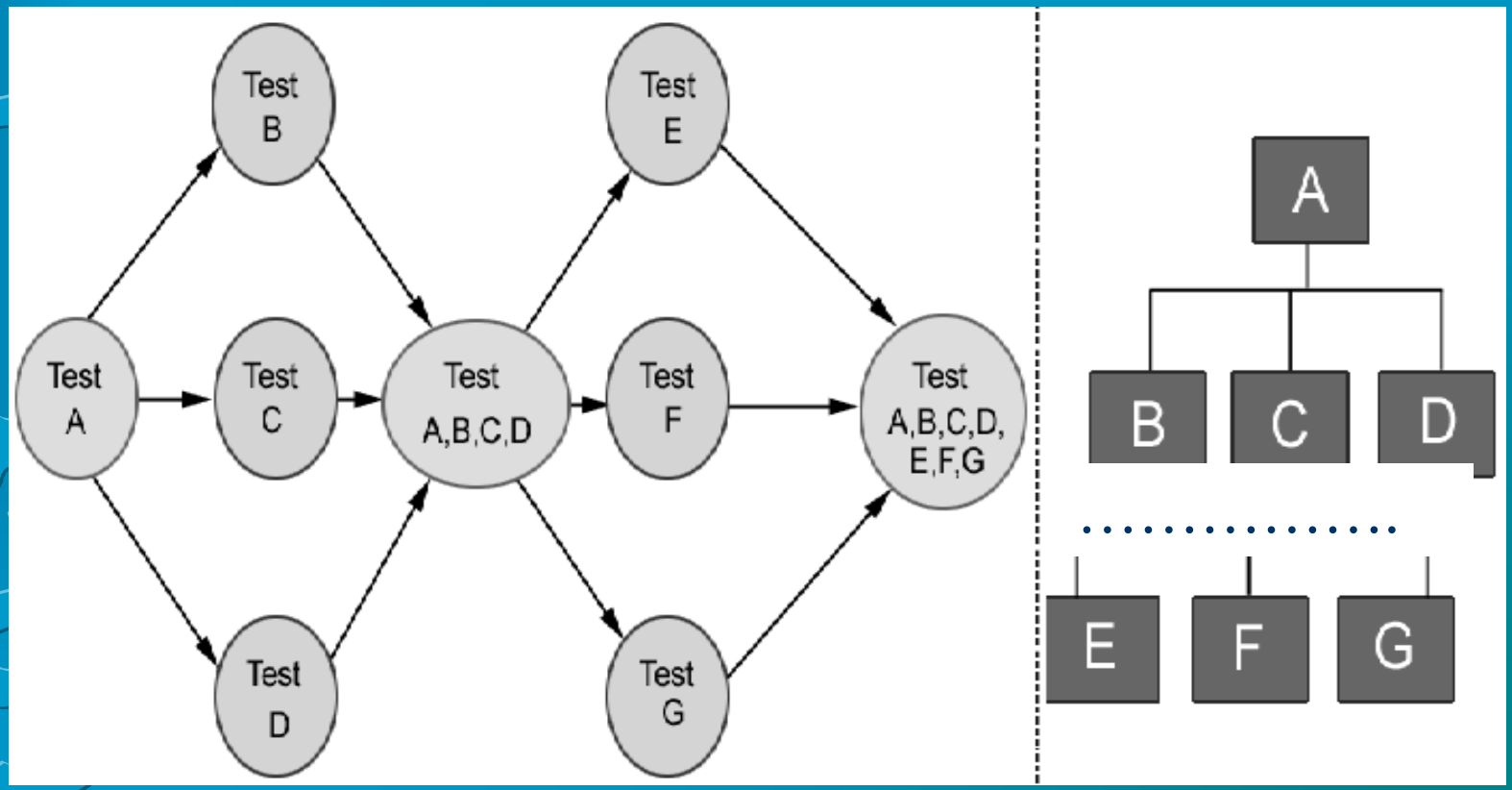


自底向上法的主要优缺点：底层模块接口错误发现早，底层关键模块错误发现早；需要驱动程序、后期验证主要功能；

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

Critical-First Integration

- Integrate the most critical components first
- Add remaining pieces later
- Issues:
  - guarantees that most important components work first
  - may be difficult to integrate

Function-at-a-Time Integration

- Integrate all modules needed to perform one function at the same time
- For each function, add another group of modules
- Issues:
  - makes for easier test generation简化测试
  - may postpone function interaction too long
    推迟功能间的交互与集成

面向对象的集成测试方法

- 由行为模型（状态、活动、顺序和合作图）导出的测试
- 状态转换图（STD）可以用来帮助导出类的动态行为的测试序列，以及这些类与之合作的类的动态行为测试序列。
- 为了说明问题，仍用前面讨论过的account类。开始由empty acct状态转换为setup acct状态。类实例的大多数行为发生在working acct状态中。而最后，取款和关闭分别使account类转换到non-working acct和dead acct状态。

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

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
- 每项功能符合实际要求
- 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

回归测试的目的

- 所做的修改达到了预定的目的，如错误得到了改正，新功能得到了实现，能够适应新的运行环境等；
- 不影响软件原有功能的正确性。

回归测试

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

回归测试

回归测试的组织和实施

- 识别修改部分
- 确定新的基线测试用例库T0
- 按照测试策略选择测试用例测试修改的软件
- 补充新的测试用例，形成新的用例库T1
- 用T1测试

冒烟测试

- 关于冒烟测试，应该是微软首先提出来的一个概念，和微软一直提倡的每日build有很密切的联系。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。

冒烟测试的说法据说是：

就象生产汽车一样，汽车生产出来以后，首先发动汽车，看汽车能否冒烟，如果能，证明汽车最起码可以开动了。说明完成了最基本的功能。

简单的说，就是先保证系统能跑的起来，不至于让测试工作做到一半突然出现错误导致业务中断。目的就是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

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)

6.4压力测试、容量测试和性能测试

压力测试、容量测试和性能测试的测试目的虽然有所不同，但其手段和方法在一定程度上比较相似，通常会使用特定的测试工具，来模拟超常的数据量、负载等，监测系统的各项性能指标，如CPU和内存的使用情况、响应时间、数据传输量等。

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

- 稳定性测试，也叫可靠性测试（reliability testing），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
  - 稳定性测试的方法也很简单，即采用24*7（24小时*7天）的方式让系统不间断运行，至于具体运行多少天，是一周还是一个月，视项目的实际情况而定。
  - 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

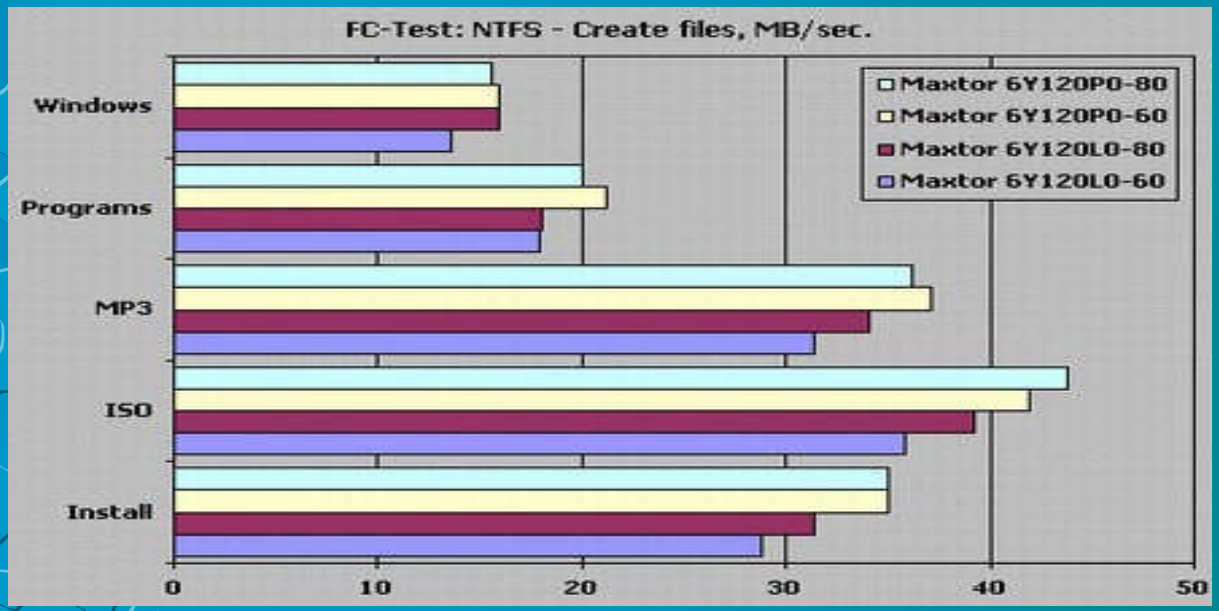
压力测试

- 破坏性加压测试：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）。 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，什么压力是系统的极限
- 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

压力测试、容量测试和性能测试

- 压力测试、容量测试和性能测试的测试目的虽然有所不同，但其手段和方法在一定程度上比较相似，通常会使用特定的测试工具，来模拟超常的数据量、负载等，监测系统的各项性能指标，如CPU和内存的使用情况、响应时间、数据传输量等。

- 我们可以通过一次测试达到三者的目的

性能测试

- 业界通常将三者泛称为性能测试
- 目的：通过性能测试找出系统的性能瓶颈，解决问题，提升性能，并且给出性能指标数据。最大限度满足用户需求，提升产品质量

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

主题内容

- 系统性能与性能测试
- 用户需求与测试目标
- 系统性能测试与瓶颈分析关键步骤
- 系统性能测试与瓶颈分析工具
- 系统性能测试与瓶颈分析实例

实例分析

网络配置

- 总部的连接方式采用100M局域网
- 部门及项目点与总部的通讯采用多种连接方式，包括宽带、ISDN、企业骨干网以及拨号等

实例分析

系统业务部署规模

该系统并发用户数为100，业务涉及总部、下属部门和项目点，每个下属部门和项目点各有2个连接点，其中下属部门连接数为15×2(目前应用为10个下属部门，考虑未来业务扩容的需求为15个)，项目点接数为35×2(目前应用为23个项目点，考虑未来业务扩容的需求为35个)。

实例分析

设计测试用例

用例名称		用例中事务	并发用户数	数据量	网络环境(带宽)
制度文档	信息上传	1. 查询所有 2. 信息上传	20、50…100	查询库数据量满足： 5000条、10000条、50000条、100000条	100M局域网 10M局域网 2M企业内部网 56Kbps Modem
	附件上传下载	1. 查询所有 2. 附件上传 3. 附件下载	20、50…100	查询库数据量满足： 5000条、10000条、50000条、100000条 上传下载附件满足： 附件大小为200k 附件大小为1M 附件大小为5M	
项目管理		1. 项目选择 2. 新增项目 3. 条件查询	20、50…100	查询库数据量满足： 1000条、5000条、10000条、100000条	
工作记事		1. 新增记事 2. 条件查询	20、50…100	查询库数据量满足： 10000条、50000条、100000条	

实例分析

客户端事务执行结果

用例名称	并发用户数	平均响应时间(s)			最大通过率(tran/s)		
		项目选择	新增项目	条件查询	项目选择	新增项目	条件查询
项目管理	40	34.451	69.669	7.545	1.5	1.125	1.875
	50	**46.552**	**86.105**	10.992	2.75	1.875	2.333

实例分析

非客户端：服务器资源

用例名称	并发用户数	CPU指标 CPU（%）		内存指标 Page In/Page Out（num/s）		硬盘指标 Disk Traffic（bytes/s）	
		CM服务器	Portal服务器	CM服务器	Portal服务器	CM服务器	Portal服务器
项目管理	50	平均值 17.281 登录阶段 68.158	平均值 84.511 最大值 100	0/3.397	0.267/74.5	4.735	5.99

实例分析　瓶颈分析—服务器资源使用瓶颈

- Portal服务器处理速度慢，出现瓶颈；CM服务器处理速度快，经常处于空闲状态。
- 负载压力期间Portal服务器CPU资源占用量大。
- 首页加载7个频道期间，CM 服务器CPU资源占用量达到100%。
- Portal服务器内存占用在业务执行结束后长时间不完全释放，并且随系统承受负载的持续，有递增趋势。
- 当系统负载压力递增时，数据库有失败SQL调用存在，造成客户端事务失败等。

实例分析　性能调优建议—系统性能调优措施

- CM、Portal、websphere、DB2的调优
- 调整Portal与CM的业务逻辑划分
- 负载压力期间Portal服务器的资源占用
- 数据库和中间件的最大连接数匹配关系
- 源代码在开发过程中的优化
- Potal服务器的网络发送时间
- 数据发送端的有关TCP重传参数设置
- 应用程序在客户端与Portal服务器之间会话的往返行程次数

实例分析

调优后效果

- 经过软件调优，服务器性能得到很大提升

  例如：并发用户从50增加到100

  响应时间从90秒降为10秒

  资源合理使用

- 经过网络调优，广域网用户的使用性能得到保证
- 在软件与网络调优的基础上，准确评估与预测系统硬件选型需求

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。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，此时非法侵入者无利可图**。（如：防盗门）

系统级别安全性：具备系统访问权限的才能访问系统

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度。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。理论上说，可靠的软件系统应该是正确、完整、一致和健壮的。

- **规定的时间**：软件系统运行后工作与挂起的累计时间
- **规定的环境条件:**规定软件系统运行时计算机的配置情况以及对输入数据的要求，判断责任方
- **规定的功能:** 明确要测试的功能与任务，要保证所规定的功能的可靠性

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。容错性测试包括两个方面：

- 输入异常数据或进行异常操作，以检验系统的保护性。如果系统的容错性好的话，**系统只给出提示或内部消化掉**，而不会导致系统出错甚至崩溃。
- 灾难恢复性测试。通过各种手段，让软件强制性地发生故障，然后验证系统已保存的用户**数据是否丢失、系统和数据是否能尽快恢复**。

目标：应用程序、数据库和系统应该在恢复过程完成时立即返回到已知的预期状态。字段、数据等

参见书P125：目标、测试范围、完成标准、需考虑的特殊事项

软件测试与维护

测试环境的建立

测试环境与辅测试环境

软件环境分为主测试环境和辅测试环境。

主测试环境是测试软件功能、安全可靠性、性能、易用性等大多数指标的主要环境

辅助测试环境满足特殊的测试需求

- 兼容性测试
- 模拟真实环境测试
- 横向对比测试

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

构建软件测试环境

- 软件测试环境就象是一个舞台，可让所有的被测软件在这个舞台上各显其能，尽情“表演”，而我们的测试工程师们就像是一个个评委，对每个被测软件的“表演”打分、评判。因而，软件测试环境构建的是否合理、稳定和具有代表性，将直接影响到软件测试结果的真实性、可靠性和正确性，所以千万不可小窥了软件测试环境的搭建工作，它是测试实施的一个重要阶段和环节，其重要性是不言而喻的；
- 另一方面，不同（版本）的操作系统、不同（版本）的数据库，不同（版本）的网络服务器、应用服务器，再加上不同的系统架构等的组合，使得要构建的软件测试环境多种多样、不胜枚举；
- 而且现在随着软件运行环境的多样性、配置各种相关参数的“浩大工程”和测试软件的兼容性等方面的需要，使得构建软件测试环境的工作变得较为复杂和频繁，

13.3.2 选择和规划实验室

一旦确定有建立测试实验室的必要，就需要为实验室选择场所并规划它的配置。应当考虑各种因素，例如空间尺寸、照明、布局、功能区、温度、湿度、放火和安全、电源、静电、设施等等，尽可能描绘出实验室的量化层平面图进行规划、不断地完善调整规划。

运作之测试部门建设方案 模式

- 独立运作
- 与第三方测试机构合作
- 与相关企业合作
- 测试外包
- 项目经理式外包

……