软件测试

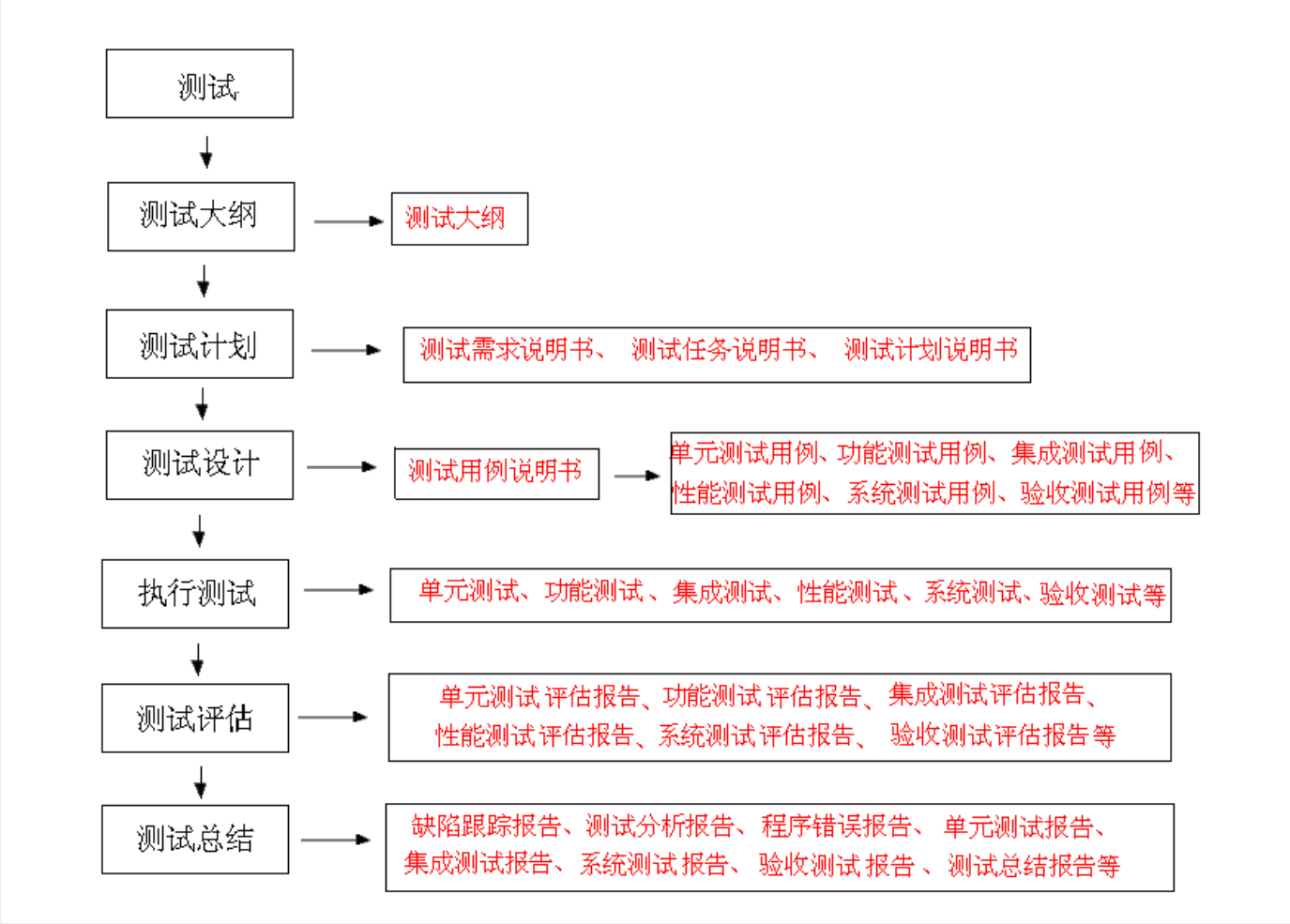
张程
Email：bootan@cqu.edu.cn
QQ:80463125

- 基本测试过程
  - *基本测试过程原则：尽早测试、经常测试、充分测试。*
  - *开发过程与测试过程：分析、测试、设计、测试、编码、测试 。*
  - *测试计划应该是按照开发者的要求并用具体例子来描述一个测试计划的层次结构以及各个测试计划相联系的标准模版*
- 测试的五个问题
  - *谁执行了测试?*
  - *测试什么？*
  - *什么时候测试?*
  - *怎样测试?*
  - *测试应进行到何种程度?*

阶　段	输入和要求	输出
需求分析阶段	市场/产品需求定义、分析文档和相关技术文档 要求：需求定义要准确、完整和一致，真正理解客户的需求	需求定义中问题列表，批准的需求分析文档 测试计划书的起草
设计阶段	产品规格设计说明、系统架构和技术设计文档、测试计划和测试用例 要求：系统结构的合理性、处理过程的正确性、数据库的规范化、模块的独立性等 清楚定义测试计划的策略、范围、资源和风险，测试用例的有效性和完备性	设计问题列表、批准的各类设计文档、系统和功能的测试计划和测试用例 测试环境的准备
单元测试阶段	源程序、编程规范、产品规格设计说明书和详细的程序设计文档 要求：遵守规范、模块的内聚性、功能实现的一致性和正确性	缺陷报告、跟踪报告；完善的测试用例、测试计划； 对系统功能及其实现等了解清楚； 获得可组装的单元；
集成测试阶段	通过单元测试的模块或组件、编程规范、集成测试规格说明和程序设计文档、系统设计文档 要求：接口定义清楚且正确、模块或组件一起工作正常、能集成为完整的系统	缺陷报告、跟踪报告；完善的测试用例、测试计划；集成测试分析报告； 集成后的系统

阶　段	输入和要求	输出
功能验证阶段	代码软件包（含文档），功能详细设计说明书；测试计划和用例 要求：模块集成 功能的正确性、适用性	缺陷报告、代码完成状态报告、功能验证测试报告
系统测试阶段	修改后的软件包、测试环境、系统测试用例和测试计划 要求：系统能正常地、有效的运行，包括性能、可靠性、安全性、兼容性等。	缺陷报告、系统性能分析报告、缺陷状态报告、阶段性测试报告
验收测试阶段	产品规格设计说明、预发布的软件包、确认测试用例 要求：向用户表明系统能够按照预定要求那样工作，使系统最终可以正式发布或向用户提供服务。用户要参与验收测试	用户验收报告、缺陷报告审查、版本审查 最终测试报告
维护	变更的需求、修改的软件包、测试用例和计划 要求：新的或增强的功能正常、原有的功能正常，不能出现回归缺陷	缺陷报告、更改跟踪报告、测试报告

软件测试工作


测试
测试大纲
测试大纲
测试计划
测试需求说明书、 测试任务说明书、 测试计划说明书
测试设计
测试用例说明书
单元测试用例、功能测试用例、集成测试用例、
性能测试用例、系统测试用例、验收测试用例等
执行测试
单元测试、功能测试 、集成测试、性能测试 、系统测试、验收测试等
测试评估
单元测试 评估报告、功能测试 评估报告、集成测试评估报告、
性能测试 评估报告、系统测试 评估报告、 验收测试评估报告等
测试总结
缺陷跟踪报告、测试分析报告、程序错误报告、 单元测试报告、
集成测试报告、系统测试 报告、 验收测试 报告 、测试总结报告等

单元测试

概述

- 单元：一个最小的单元应该有明确的功能、性能、接口定义而且可以清析地与其他单元区分开
- 单元测试（Unite Testing）又称为模块测试，主要来检验软件设计中最小的单位——模块。一般来说模块的内聚程度高，每一个模块只能完成一种功能，因此模块测试的程序规模小，易检查出错误。我们可以通过单元测试进行程序语法检查和程序逻辑检查，验证程序的正确性。单元测试非常重要，因为它影响的范围比较广，主要表现在如果一个单元模块的一个函数或者参数出现问题，会造成后面很多问题的出现，而且如果单元测试做不好，使得集成测试或者后面系统测试工作也做不好。做好单元测试是一个重要而且基础性的工作，主要的测试方法分为人工测试和自动化测试两种方式
- 单元测试目的：
  - *检查单元模块内部的错误，为软件的评审验收提供依据。*
  - *单元测试是以程序设计说明书和之前所作的测试数据（正常的和错误的）为指导，测试模块内重要的路径，以检查出错误；*
  - *检验信息能否正确地流入和流出单元；*
  - *在单元测试工作过程中，其内部数据能否保持其完整性，包括内部数据的形式、内容及相互关系不发生错误，也包括全局变量在单元中的处理和影响。*
  - *在为限制数据加工而设置的边界处，能否正确工作。*
  - *单元的运行能否做到满足特定的逻辑覆盖。*
  - *单元中发生了错误，其中的出错处理措施是否有效*

单元测试的内容

- 目标：确保模块被正确地编码；
- 依据：详细设计说明书、源程序；
- 过程：包括设计、脚本开发、执行、调试和分析结果；
- 执行者：开发人员和测试人员
- 测试方法：白盒为主，黑盒为辅
- 评估：通过所有单元测试用例，代码没有严重缺陷

单元测试的优点

- 是一种验证行为：程序中的每一项功能都是测试来验证它的正确性。
- 是一种设计技术：单元测试使我们把程序设计成易于调用和可测试的设计，是一种设计技术。
- 是一种编写文档的行为：单元测试是一种文档，它是展示函数或类如何使用的最佳文档。这份文档是可编译、可运行的，并且它保持最新，永远与代码同步。
- 具有回归性：单元测试避免了代码出现回归，编写完成之后，可以随时随地的快速运行测试。
- 保证：保证代码质量，保证代码的可维护性，保证代码的可扩展性

单元测试的停止准则

- 软件单元功能与设计需求一致;
- 软件单元接口与设计需求一致;
- 能够正确处理输入和运行中的错误;
- 在单元测试中发现的错误已经修改并通过了测试;
- 达到了相关的覆盖率的要求;
- 完成软件单元测试报告

单元测试检查表

- 借助单元测试检查表进行评估。

案例：

单元测试检查表

单元名称_____ 系统 _____ 构造_____

任务编号_____ 初次测试日期_____

关键测试项是否已纠正

1. 有无任何输入参数没有使用？有无任何输出参数没有产生？
2. 有无任何数据类型不正确或不一致？
3. 有无任何算法与PDL或功能需求中的描述不一致？
4. 有无任何局部变量使用前没有初始化？
5. 有无任何外部接口编码错误？即调用语句、文件存取、 数据库错误。
6. 有无任何逻辑路径错误？
7. 该单元是否有多个入口或多个正常的出口？

单元测试检查表（续）

额外测试项

8. 该单元中有任何地方与PDL与PROLOG中的描述不一致？
9. 代码中有无任何偏离本项目标准的地方？
10. 代码中有无任何对于用户来说不清楚的错误提示信息？
11. 如果该单元是设计为可重用的，代码中是有可能妨碍重用的地方？

采取的动作和说明

（请用单独的一页或多页。每一项动作必须指出所引用的问题。）

审查结果

1. 如果上述11个问题的答案均为"否"，那么测试通过，请在此标记并且在最后签名。
2. 如果代码存在严重的问题，例如多个关键问题的答案为"是"，那么程序编制者纠正这些错误，并且必须重新安排一次单元测试。

下一次单元测试的日期：_____

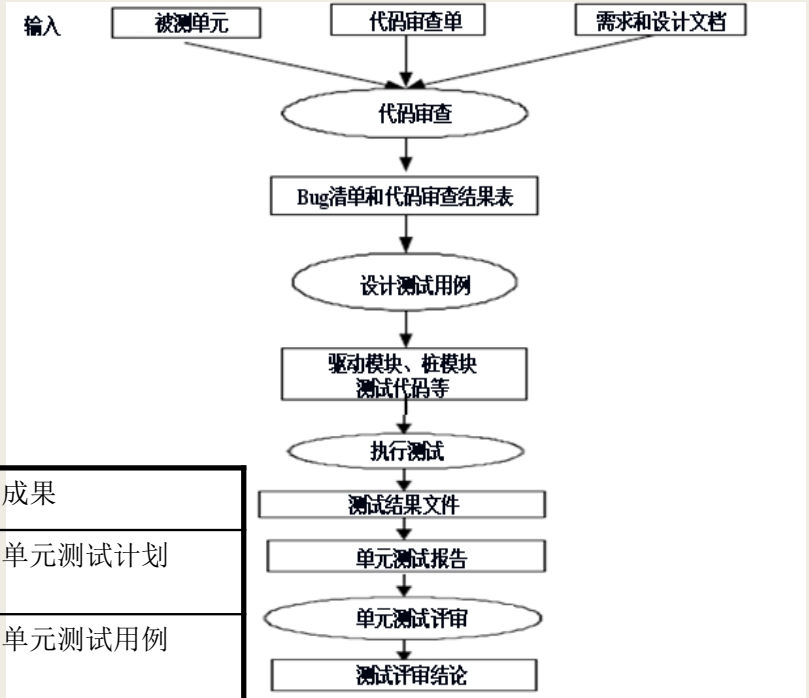
3. 如果代码存在小的缺陷，那么程序编制者纠正这些错误，并且仲裁者必须安排一次跟踪会议。

跟踪会议的日期：_____

测试人签名：_____ 日期：_____

单元测试的过程与文档管理

- 单元测试过程由以下5个步骤组成：
  - *在详细设计阶段完成单元测试计划；*
  - *建立单元测试环境，完成测试设计和开发；*
  - *执行单元测试用例，并详细记录测试结果；*
  - *判定测试用例是否通过*
  - *提交《单元测试报告》*



	时间	依据	任务	成果
计划阶段	详细设计阶段后	《软件需求规格说明书》 《详细设计说明书》	制定测试计划	单元测试计划
设计阶段	《单元测试计划》提交后	《单元测试计划》《软件详细设计说明书》	测试用例的编写 驱动模块、桩模块的设计	单元测试用例
执行阶段	编码完成后	《单元测试用例》《软件需求规格说明书》《详细设计说明书》	执行测试用例 记录缺陷	《缺陷跟踪报告》
评估阶段		《单元测试用例》《缺陷跟踪报告》《缺陷检查表》	完备性评估 代码覆盖率评估	《单元测试报告》

单元测试的任务

- 模块独立执行通路测试：检查每一条独立执行路径的测试。保证每条语句被至少执行一次。
  - *误解或用错了算符优先级。*
  - *混合类型运算。*
  - *变量初值错。*
  - *精度不够。*
  - *表达式符号错。*
  - *其它*
- 模块局部数据结构测试：检查局部数据结构完整性
  - *不适合或不相容的类型说明。*
  - *变量无初值。*
  - *变量初始化或默认值有错。*
  - *不正确的变量名或从来未被使用过。*
  - *出现上溢或下溢和地址异常。*
  - *其它*

单元测试的任务（续）

- 模块接口测试：检查模块接口是否正确
  - *输入的实际参数与形式参数是否一致。*
    - 个数、属性、量纲
  - *调用其他模块的实际参数与被调模块的形参是否一致。*
    - 个数、属性、量纲
  - *调用预定义函数时所用参数是否一致。*
    - 个数、属性、量纲
  - *全程变量的定义在各模块是否一致。*
  - *外部输入、输出*
    - 文件、缓冲区、错误处理
  - *其它*
- 模块边界条件测试：检查临界数据处理的正确性
  - *普通合法数据的处理。*
  - *普通非法数据的处理。*
  - *边界值内合法边界数据的处理。*
  - *边界值外非法边界数据的处理。*
  - *其它*

单元测试的任务（续II）

- 模块的各条错误处理通路测试：预见、预设的各种出错处理是否正确有效。
  - *输出的出错信息难以理解。*
  - *记录的错误与实际不相符。*
  - *程序定义的出错处理前系统已介入。*
  - *异常处理不当。*
  - *未提供足够的定位出错的信息。*
  - *其它*

集成测试

基本概念

- 集成
  - *把多个单元组合起来形成更大的单元*
- 集成测试
  - *也称为组装测试、联合测试、子系统测试、部件测试*
  - *在假定各个软件单元已经通过了单元测试的前提下，检查各个软件单元接口之间的协同工作（参数、全局数据结构、文件、数据库）是否正确。*
  - *通常采用黑盒测试用例设计方法、灰盒测试方法*
- 集成测试关注的问题
  - *模块间的数据传递是否正确？（参数个数、类型、次序）*
  - *一个模块的功能是否会对另一个模块的功能产生错误的影响？（资源内容及操作）*
  - *全局数据结构是否有问题，会不会被异常修改？（使用一致性）*
  - *集成后各个模块的累积误差是否会扩大，是否达到不可接受的程度？（精度缺失、逻辑错误）*
  - *模块组合起来的功能能否满足要求？（需求覆盖）*

概述

- 模块分析
  - *集成测试的第一步，也是最重要的工作之一。*
  - *软件工程有一条事实上的原则，即2/8原则，即测试中发现的80%的错误可能源于20%的模块。例如，IBM OS/370操作系统中，用户发现的47%的错误源于4%的模块。*
  - *一般将模块划分为3个等级*
    - 高危模块、一般模块和低危模块
  - *高危模块应该优先测试*
- 模块的划分常常遵循下列几个原则
  - *本次测试希望测试哪个模块*
  - *把与该模块最紧密的模块聚集在一起*
  - *在考虑划分后的外围模块，并分析外围模块和被集成模块之间的信息流是否容易模拟和控制*

概述（续）

- 集成测试与系统测试的区别

测试对象	单元	系统
测试时间	开发过程	开发完成
测试方法	黑白结合	黑盒
测试内容	接口	需求
测试目的	接口错误	需求不一致
测试角度	开发者	用户

概述（续）

- 集成测试与开发的关系
  - *集成测试与软件开发过程中的概要设计阶段相对应，软件概要设计中关于整个系统的体系结构是集成测试用例输入的基础。*
    - 概要设计作为软件设计的骨架，从一个成熟的体系结构中，可以清晰地看出大型系统中的组件或子系统的层次构造。
    - 为集成测试策略的选取提供了重要的参考依据，从而可以减少集成测试过程中桩模块和驱动模块开发的工作量，促使集成测试快速、高质量的完成
- 集成测试的层次
  - *传统软件按集成粒度不同，可以分为4个层次*
  - *模块内集成测试*
  - *模块间集成测试*
  - *子系统内集成测试*
  - *子系统间集成测试*
  - *面向对象的应用系统，按集成粒度不同可分为2个层次*
  - *类内集成测试*
  - *类间集成测试*

概述（续）

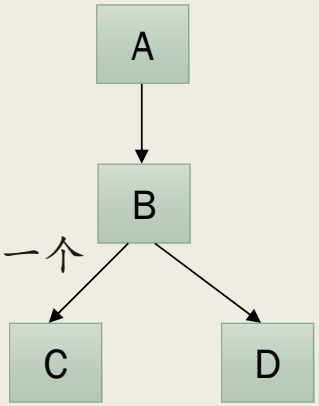
- 集成测试的原则
  - *所有公共接口必须被测试到*
  - *关键模块必须进行充分测试*
  - *集成测试应当按一定层次进行*
  - *集成测试策略选择应当综合考虑质量、成本和进度三者的关系*
  - *集成测试应当尽早开始，并以文档为基础*
  - *在模块和接口的划分上，测试人员应该和开发人员进行充分沟通*
  - *当测试计划中的结束标准满足时，集成测试才能结束*
  - *当接口发生修改时，涉及到的相关接口都必须进行回归测试*
  - *集成测试应根据集成测试计划和方案进行，不能随意测试*
  - *项目管理者应保证测试用例经过审核*
  - *测试执行结果应当如实的记录*

集成测试重点考虑的内容

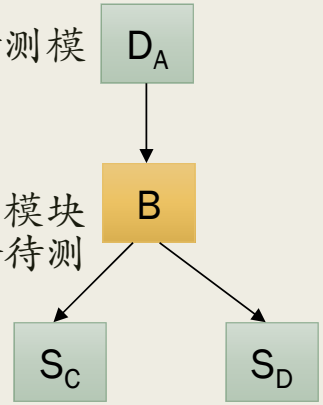
- 采用何种集成方法来进行集成测试；
- 集成测试过程中连接各个模块的顺序；
- 模块代码编制和测试进度是否与集成测试的顺序一致；
- 模块之间的接口有没有错误；
- 测试过程中是否需要专门的硬件设备；
- 在把各个模块连接起来的时候，穿越模块接口的数据是否会丢失；
- 功能有没有达到预期效果；
- 模块相互调用时有没有引入了新的问题；
- 计算的误差累计达到了不能接受的程度；
- 模块组合能否正常工作；
- 各个子功能组合起来，能否达到预期要求的父功能；
- 一个模块的功能是否会对另一个模块的功能产生不利的影响；
- 进行回归测试，以保证不引入新的错误。
- 数据经过接口可能丢失；
- 全局数据结构是否有问题

集成测试策略

- 模块组装方法
  - *非渐增式集成*
    - 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
  - *渐增式集成*
    - 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，然后再把下一个待测试的模块结合起来进行测试
    - 同时完成单元测试和集成测试

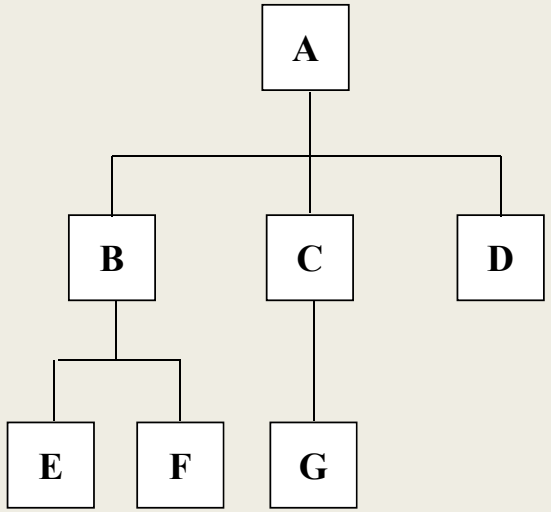


- 辅助模块
  - *驱动模块（Driver）*
    - 用以模拟待测模块的上级模块；接受测试数据，并传送给待测模块，启动待测模块，并打印出相应的结果。
  - *桩模块（Stub）*
    - 也称存根程序。用以模拟待测模块工作过程中所调用的模块。桩模块由待测模块调用，它们一般只进行很少的数据处理，例如打印入口和返回，以便于检验待测模块与其下级模块的接口。

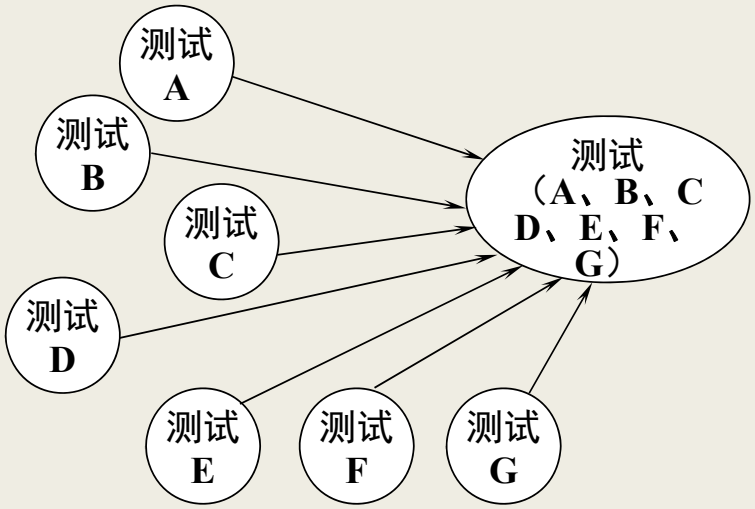


集成测试策略

- 非渐增式集成
  - *首先对每个子模块进行测试（即单元测试），然后将所有模块全部集成起来一次性进行集成测试。*
  - *举例：*



程序结构图



非渐增式集成

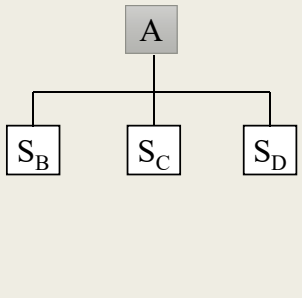
集成测试策略

- 渐增式集成
  - *与“一步到位”的非渐增式集成相反，它把程序划分成小段来构造和测试。*
  - *容易定位和改正错误；对接口可以进行更彻底的测试；可以使用系统化的测试方法。目前在进行集成测试时普遍采用渐增式集成方法。*
  - *渐增方式有自顶向下和自底向上两种集成策略*

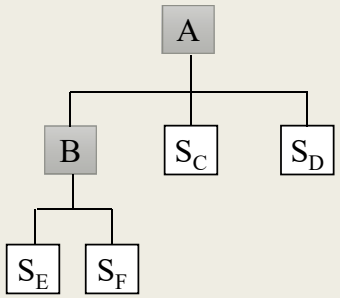
集成测试策略-自顶向下集成

- 自顶向下集成
  - *从主控制模块开始，沿着程序的控制层次向下移动，逐渐把各个模块结合起来。在把附属于（及最终附属于）主控制模块的那些模块组装到程序结构中去。*
  - *使用深度优先的策略，或者使用宽度优先的策略*
- 自顶向下集成测试的步骤
  - *对主控制模块进行测试，测试时用桩模块代替所有直接附属于主控制模块的模块。*
  - *根据选定的结合策略（深度优先或宽度优先），每次用一个实际模块代换一个桩模块（新结合进来的模块往往又需要新的桩模块）。*
  - *在结合进一个模块的同时进行测试。*
  - *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（即，全部或部分地重复以前做过的测试）。*

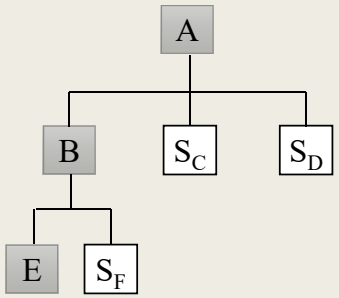
集成测试策略-自顶向下集成



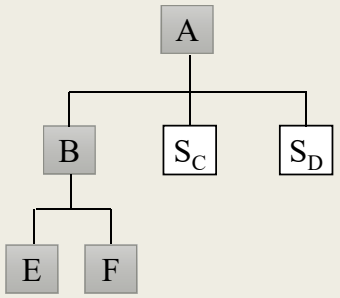
（a）测试A



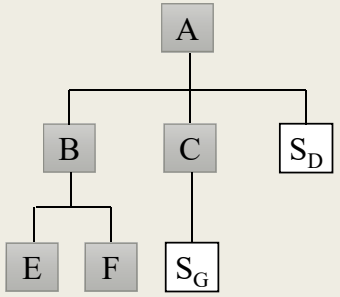
（b）测试B



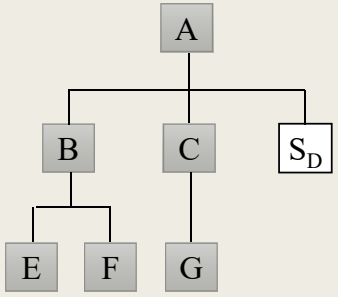
（c）测试E



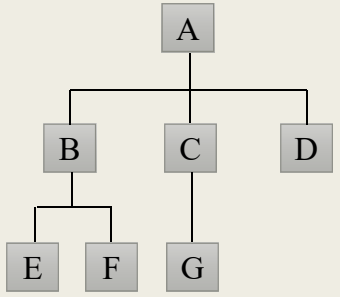
（d）测试F



（e）测试C



（f）测试G



（g）测试D

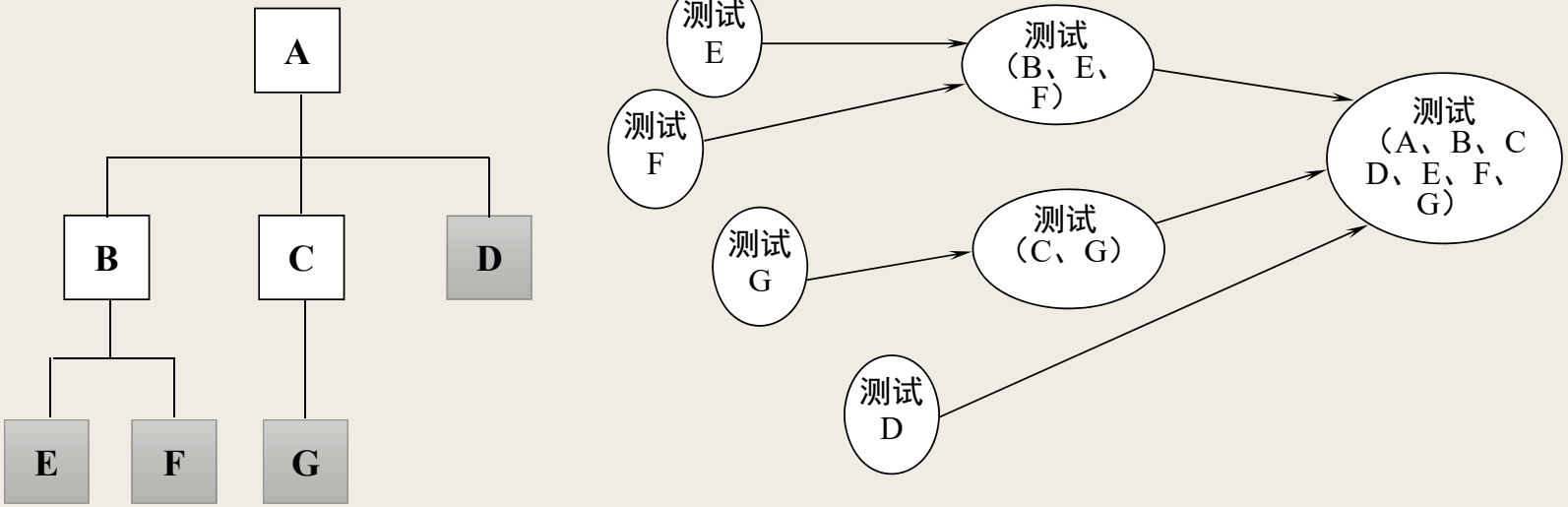
集成测试策略-自顶向下集成

- 优点
  - *可以及早发现主控模块的问题解决*
  - *如果选择深度优先的结合方法，可以在早期实现并验证一个完整的功能，增强开发人员和用户双方的信心*
- 缺点
  - *桩模块代替了低层次的实际模块，没有重要的数据自下往上流*

集成测试策略-自底向上集成

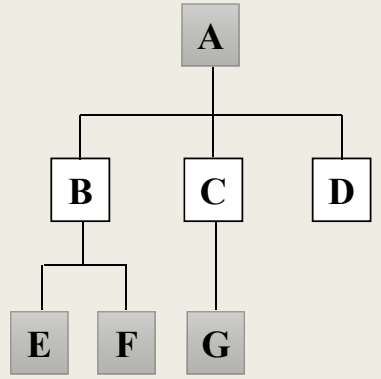
- 自底向上集成
  - *从“原子”模块（即最底层的模块）开始组装和测试*
  - *不需要桩模块*
  - *步骤*
    - 把低层模块组合成实现某个特定的软件子功能的族
    - 写一个驱动程序（用于测试的控制程序），协调测试数据的输入和输出
    - 对由模块组成的子功能族进行测试
    - 去掉驱动程序，沿软件结构自下向上移动，把子功能族组合起来形成更大的子功能组

集成测试策略-自底向上集成

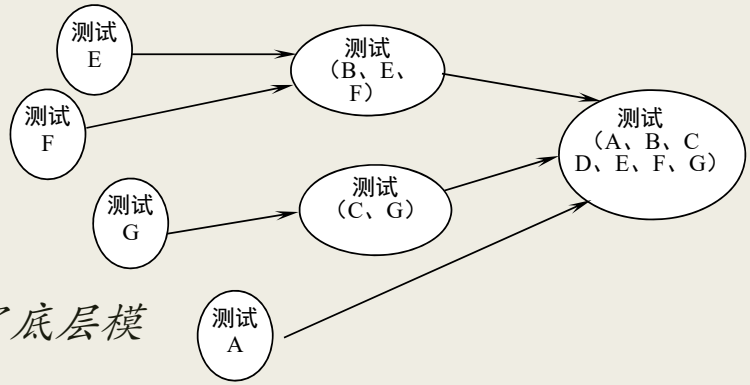


集成测试策略-三明治集成

- 三明治集成
  - *混合增量式测试策略，综合了自顶向下和自底向上两种集成方法的优点*
  - *桩模块和驱动模块的开发工作都比较小*
  - *代价：一定程度上增加了定位缺陷的难度*
- 三明治集成测试的过程
  - *确定以哪一层为界来进行集成，如图的B模块*
  - *对模块B及其所在层下面的各层使用自底向上的集成策略*
  - *对模块B所在层上面的层次使用自顶向下的集成策略*
  - *对模块B所在层各模块同相应的下层集成*
  - *对系统进行整体测试*
- 优点：
  - *将自顶向下和自底向上的集成方法有机地结合起来*
  - *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*
- 主要缺点：在真正集成之前每一个独立的模块没有完全测试过



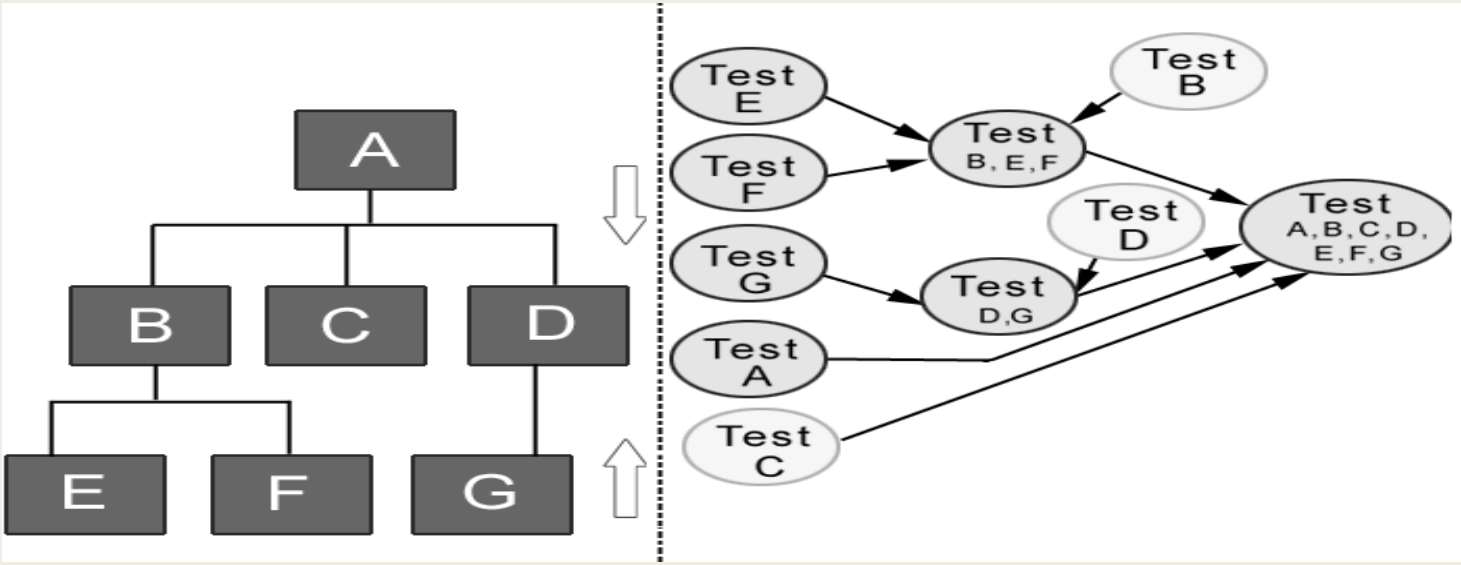
程序结构图



三明治集成

集成测试策略-改进的三明治集成

- 不仅自两头向中间集成，而且保证每个模块得到单独的测试，使测试进行得比较彻底



集成测试用例设计

- 为系统运行设计用例
  - *目的：达到合适的功能覆盖率和接口覆盖率*
  - *可使用的主要测试分析技术*
    - 等价类划分
    - 边界值分析
    - 基于决策表的测试
- 为正向集成测试设计用例
  - *测试目标：验证集成后的模块是否按照设计实现了预期的功能。*
  - *可直接根据概要设计文档导出相关测试用例，使用的主要测试分析技术有*
    - 输入域测试
    - 输出域测试
    - 等价类划分
    - 状态转换测试
    - 规范导出法

集成测试用例设计（续）

- 为逆向集成测试设计用例
  - *测试目标：分析被测接口是否实现了需求规格没有描述的功能，检查规格说明中可能出现的接口遗漏等。*
  - *可使用的主要测试分析技术*
    - 错误猜测法
    - 基于风险的测试
    - 基于故障的测试
    - 边界值分析
    - 特殊值测试
    - 状态转换测试
- 为满足特殊需求设计用例
  - *测试目标：接口的安全性指标、性能指标等*
  - *可使用的主要测试分析技术为规范导出法*
- 为覆盖设计用例
  - *可使用的主要测试分析技术有*
    - 功能覆盖分析
    - 接口覆盖分析

集成测试过程

- 集成测试可划分为5个阶段：计划阶段、设计阶段、实施阶段、执行阶段、评估阶段
- 计划阶段（制定行动指南）
  - *确定被测试对象和测试范围*
  - *评估集成测试被测试对象的数量及难度，即工作量*
  - *确定角色分工和划分工作任务*
  - *标识出测试各个阶段的时间、任务、约束条件*
  - *考虑一定的风险分析及应急计划*
  - *考虑和准备集成测试需要的测试工具、测试仪器、环境等资源*
  - *考虑外部技术支援的力度和深度，以及相关培训安排；定义测试完成标准*
- 设计阶段
  - *被测对象结构分析*
  - *集成测试模块分析*
  - *集成测试接口分析*
  - *集成测试策略分析*
  - *集成测试工具分析*
  - *集成测试环境分析*
  - *集成测试工作量估计和安排*

集成测试过程（续）

- 实施阶段
  - *前提条件：详细设计阶段的评审已经通过*
  - *环节*
    - 集成测试用例设计
    - 集成测试规程设计
    - 集成测试代码设计
    - 集成测试脚本开发
    - 集成测试工具开发或选择
- 执行阶段
  - *按照相应的测试规程，借助集成测试工具，并把需求规格说明书、概要设计、集成测试计划/设计/用例/代码/脚本作为测试执行的依据来执行集成测试用例。*
  - *测试执行的前提条件就是单元测试已经通过评审。*
  - *测试执行结束后，测试人员要记录下每个测试用例执行后的结果，填写集成测试报告，最后提交给相关人员评审*
- 评估阶段
  - *集成测试执行结束后，要召集相关人员（测试设计人员、编码人员、系统设计人员等）对测试结果进行评估，确定是否通过集成测试*

面向对象的集成测试

- 面向对象的程序是由若干对象组成的，对象的协作方式决定了程序能做什么，从而决定了这个程序执行的正确性。
- 交互测试的重点：确保对象的消息传送能够正确进行
- 交互测试的执行环境
  - *嵌入到应用程序中的交互对象*
  - *在独立测试工具提供的环境中*

面向对象的集成测试-对象交互

- 对象交互定义
  - *一个对象（发送者）对另一个对象（接收者）的请求，发送者请求接收者执行接收者的一个操作，而接收者进行的所有处理工作就是完成这个请求。*
  - *包含了对象及其组件的消息，还包含了对象和与之相关的其他对象之间的消息。*
- 对象交互的测试方法分类
  - *原始类*
    - 使用类的单元测试方法
  - *汇集类*
  - *协作类*

面向对象的集成测试-对象交互

- 汇集类测试
  - *汇集类的特征*
    - 汇集类在它们的说明中使用对象，但从不请求这些对象的任何服务
    - 存放这些对象的引用（或指针），程序中常表现为对象之间一对多的关系。
    - 创建这些对象的实例。
    - 删除这些对象的实例
  - *测试方法*
    - 可以使用测试原始类的方法来测试汇集类。
    - 测试驱动程序要创建一些实例，这些实例作为消息中的参数被传递给一个正在测试的集合。
  - *测试目的*
    - 保证创建的实例被正确从集合中移出

面向对象的集成测试-对象交互

- 协作类测试
  - *协作类的定义*
    - 协作类的一个或多个操作中使用其他的对象并将其作为它们的实现中不可缺少的一部分
    - 当类接口中的一个操作的某个后置条件引用了一具对象的实例状态，并且（或者）说明那个对象的某个属性被使用或修改了，那么这个类就是一个协作类
  - *测试方法*
    - 协作类测试的复杂性远远高于汇集类或原始类的测试

面向对象集成测试的常用方法

- 抽样测试
  - *抽样测试提供了一种运算法则，它使我们能够从一组可能的测试用例中选择一个测试序列。*
  - *不要求一定要首先明确如何来确定测试用例的总体。*
  - *测试过程的目的在于定义感兴趣的测试总体，然后定义一种方法，以便在这些测试用例中选择哪些被构建、哪些被执行*
- 正交阵列测试
  - *通过定义一组交互对象的配对方式组合，以尽力限制测试配置的组合数目激增。*
  - *正交阵列是一个数值矩阵，其中的每一列代表一个因素，即实验中的一个变量。在一个正交阵列中将各个因素组合成配对情况。*
  - *3个因素（每个因素有3个层次）配对方式的组合情况*

	1	2	3	4	5	6	7	8	9
A	1	1	1	2	2	2	3	3	3
B	1	2	3	1	2	3	1	2	3
C	3	2	1	2	1	3	1	3	2

分布式对象测试

- 分布式对象的概念和特点
  - *分布式对象的概念*
    - 线程是一个操作系统进程内能够独立运行的内容，它拥有自己的计算器和本地数据。
    - 线程是能够被调度执行的最小单位。
  - *分布式对象的特点：不确定性*
  - *适用技术*
    - 在类的层次上进行更彻底的测试
    - 在记录事件发生顺序的同时，执行大量的测试用例
    - 指定标准的测试环境
- 测试中需要注意的情况
  - *局部故障*
  - *超时*
  - *结构的动态性*
  - *线程*
  - *同步*

功能测试

功能测试

- 功能测试属于黑盒测试技术范畴，是系统测试中要进行的最基本的测试，它不用考虑软件内部的具体实现过程。
- 主要是根据产品的需求规格说明书和测试需求列表，验证产品是否符合产品的需求规格。
- 需求规格说明是功能测试的基本输入。因此先对需求规格进行分析，明确功能测试的重点。
- 步骤进行：
  - *为所有的功能需求（其中包括隐含的功能需求）加以标识；*
  - *为所有可能出现的功能异常进行分类分析并加以标识；*
  - *对前面表示的功能需求确定优先级。*
  - *对每个功能进行测试分析，分析其是否可测、采用何种测试方法、测试的入口条件、可能的输入、预期输出等等。*
  - *是否需要开发脚本或借助工具录制脚本。*
  - *确定要对哪些测试使用自动化测试，对哪些测试使用手工测试*

功能测试的主要内容

- 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，对异常数据的输入有提示、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

回归测试

- 回归测试是指修改了旧代码后，重新进行测试以确认修改没有引入新的错误或导致其它代码产生错误
- 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*
- 回归测试的方法
  - *再测试全部用例*
  - *基于风险选择测试*
  - *基于操作剖面选择测试*
  - *再测试修改的部分*
- *测试阶段*
  - *任一阶段*

回归测试的组织和实施

- 通过代码相依分析，识别软件中被修改的部分
- 从原有测试用例库中，排除不适用的测试用例，建立新的测试用例基线库T0
- 基于风险和操作剖面选择相结合，从新的测试用例基线库中选择测试用例构造有效的套件，测试被修改的软件
- 若回归测试套件达不到所需的覆盖要求，必须补充新的测试用例，则生成新的测试用例集T1
- 用T1测试修改后的软件

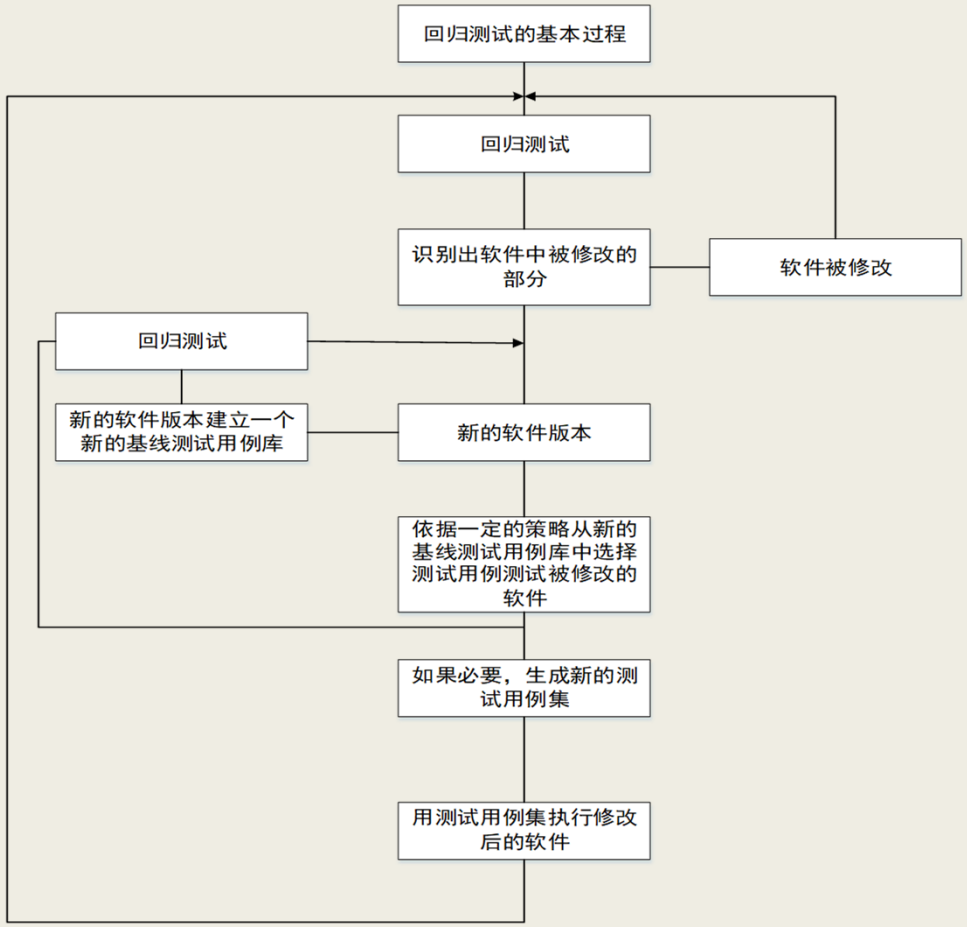
回归测试

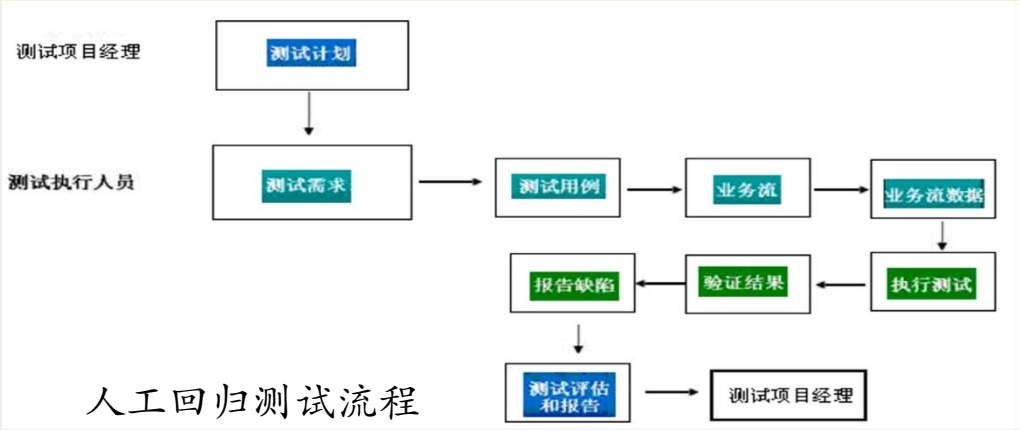
- 回归测试集
  - *能够测试软件的所有功能的代表性测试用例*
  - *专门针对可能会被修改影响的软件功能的附加测试*
  - *针对修改过的软件成分的测试*
- 回归测试的范围
  - *测试所有修改或修正过的功能模块*
  - *测试与被修改的模块相关的模块*
  - *测试所有新增加的功能模块*
  - *测试整个系统*

回归用例的选择

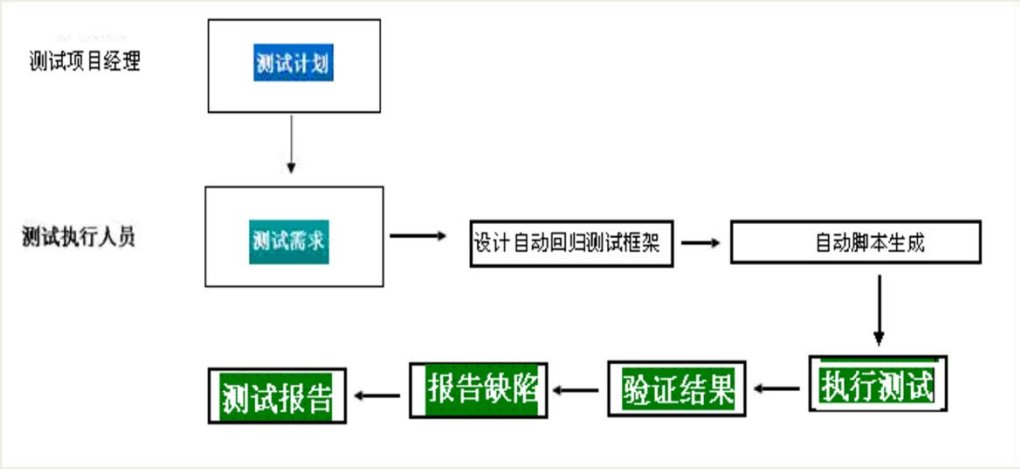
- 再测试全部用例
  - *选择测试用例库中的全部测试用例组成回归测试包。*
  - *优点：安全，最低遗漏回归错误风险*
  - *缺点：测试成本最高、工作量大、时间进度受影响*
- 基于风险选择测试
  - *基于一定的风险标准从测试用例库中选择回归测试包。*
  - *首先运行最重要的、关键的 和可疑的测试， 而跳过那些非关键的、优先级别低的或者高稳定的测试用例*
- 基于操作剖面选择测试
  - *用例分布情况反映了系统实际使用情况。*
  - *优先选择那些针对最重要或最频繁使用功能的用例，释 放和缓解最高级别的风险，有助于尽早发现那些对可靠性有最大影响的故障*
- 再测试修改的部分
  - *当测试者对修改的局部化有足够的信心时，可以通过相依性分析识别软件的修改情况并分析修改的影响，将回归测试局限于被改变的模块和它的接口上*

回归测试的基本过程





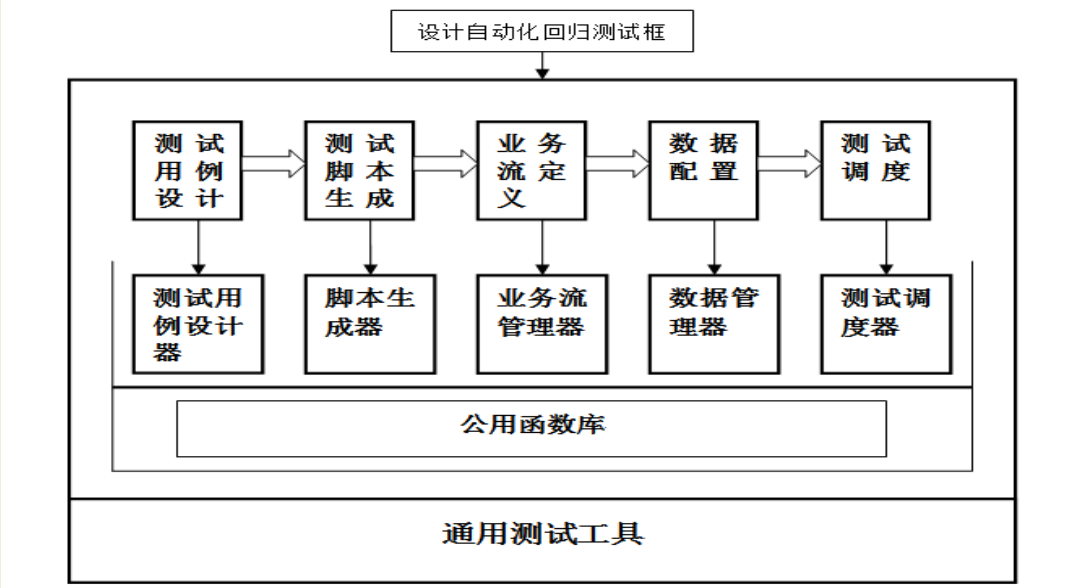
人工回归测试流程



自动化自动回归测试流程

自动回归测试框架

- 自动回归测试执行策略
  - *测试用例设计*
  - *脚本生成器*
  - *业务流管理器*
  - *数据管理器*
  - *测试调度*
- 自动回归测试框架的作用
  - *降低管理难度，使用一个管理界面，隐藏管理细节*
  - *测试设计和测试实现分离*
  - *具备自动脚本生成功能*
  - *集成数据驱动、关键字驱动和功能分解测试的最好特征*
  - *支持人工测试和自动测试*
  - *分离技术性测试人员和非技术性测试人员的工作*



自动回归测试框架的技术特点

- 基于自动测试工具的测试框架，具有测试计划驱动技术的所有优点；
- 充分利用测试工具的功能，与测试管理集成；
- 基于业务流的测试，数据也是基于业务流配置的；
- 应用与自动测试框架分开；
- 可支持技术性测试人员和非技术性测试人员使用；
- 脚本与数据分开；
- 自动测试开发与运行环境分开；
- 有专门的数据管理模块，数据维护简便；
- 自动生成代码，基本不需要编码，效率极高；
- 使用数据库，不需要人工管理大量文件；
- 采用基于组件的技术，提供预制组件；
- 支持业务功能脚本和录制脚本

回归测试克服的几个问题

- 组织回归测试时需要注意的问题
  - *各测试阶段发生的修改一定要在本测试阶段内完成回归，以免将错误遗留到下一回归测试阶段；*
  - *回归测试期间应对该软件版本冻结，将回归测试发现的问题集中修改、集中回归。*
- 重新进行回归测试时需要注意的问题
  - *安排新的测试者完成回归测试；*
  - *分配更有经验的测试者开发新的回归测试用例；*
  - *在不影响测试目标的前提下，鼓励测试者创造性地执行回归测试用例（变化的输入、按键和配置有助于激励测试者揭示新的错误）；*
  - *在回归测试当中，可以将回归测试与兼容性测试结合起来进行。*

回归测试人员应掌握的测试手段

- 要熟悉系统的业务流程，对业务需求以及相关联模块要非常清楚，保证回归测试的质量和效率；
- 及时更新和维护回归测试用例库当中的测试用例，确保执行的回归测试用例是最新的；
- 要掌握回归测试的测试用例优先级别，对优先级高的功能模块优先进行回归测试测试；
- 使用回归测试自动化工具进行测试；
- 回归测试人员应及时与开发人员进行有效的沟通，及时地反馈回归测试测试情况；
- 回归测试人员应该熟悉并掌握系统开发的各种计算机类语言

回归用例库的维护

- 软件测试项目组在进行测试的过程中会将所用到的测试用例保存到“测试用例库”中，并进行维护
- 回归测试用例库的维护方法如下：
  - *删除过时的测试用例*
  - *改进不受控的测试用例*
  - *删除冗余的测试用例*
  - *增添新的测试用例*

系统测试

系统测试

- 系统测试：已经集成好的软件系统，作为整个计算机系统的一个元素，与计算机硬件、外设、某些支持软件、数据和人员等其它系统元素结合在一起，在实际运行（使用）环境下，对计算机系统进行一系列的组装测试和确认测试
- 根本任务：证明被测系统的功能和结构的稳定性；还要有一些非功能测试：性能测试、压力测试、可靠性测试等等。
- 目的：在于通过与系统的需求定义比较，检查软件是否存在与系统定义不符合或与之矛盾的地方，以验证软件系统的功能和性能等满足其规约所指定的要求
- 系统测试属于黑盒测试范畴，不再对软件的源代码进行分析和测试
- 为什么要进行系统测试?
  - *由于软件只是计算机系统中的一个组成部分，软件开发完成之后，最终还要和系统中的硬件系统、某些支持软件、数据信息等其他部分配套运行。因此，在投入运行前要完成系统测试，以保证各组成部分不仅能单独的得到检验，而且在系统各部分协调工作的环境下也能正常工作。*

系统测试的组织和分工

- 测试组组长：组织测试；
- 测试分析员：负责设计和实现测试脚本和测试用例；
- 测试者：负责执行测试脚本中记录的测试用例。
- 同时可以邀请客户代表参与系统测试，可以与客户建立一个良好的平台，并且得到反馈信息。
- 过程：搭建好系统测试的软、硬件平台→制定软件测试计划(与开发人员多多沟通)→系统测试→提交系统测试的大量输出的拷贝文档(包括测试结果记录表格、系统测试日志和全面的系统测试总结报告)

系统测试方法

- 用户界面测试
- 恢复测试
- 安全测试
- 性能测试
- 强度测试
- 容量测试
- 配置测试
- 正确性测试
- 可靠性测试
- 兼容性测试
- Web测试
- 文档测试

用户界面测试

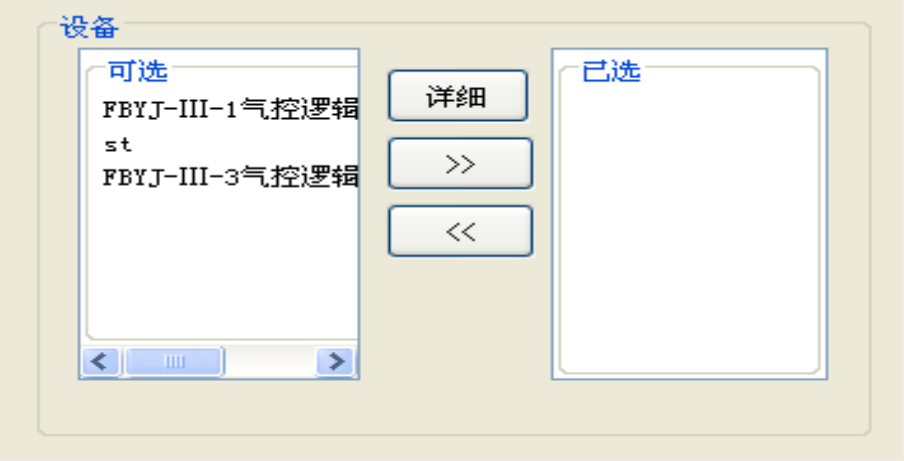
- 优秀UI应具备的7要素：
  - *符合标准和规范*
  - *一致性*
  - *正确性*
  - *直观性*
  - *灵活性*
  - *舒适性*
  - *宽容性*

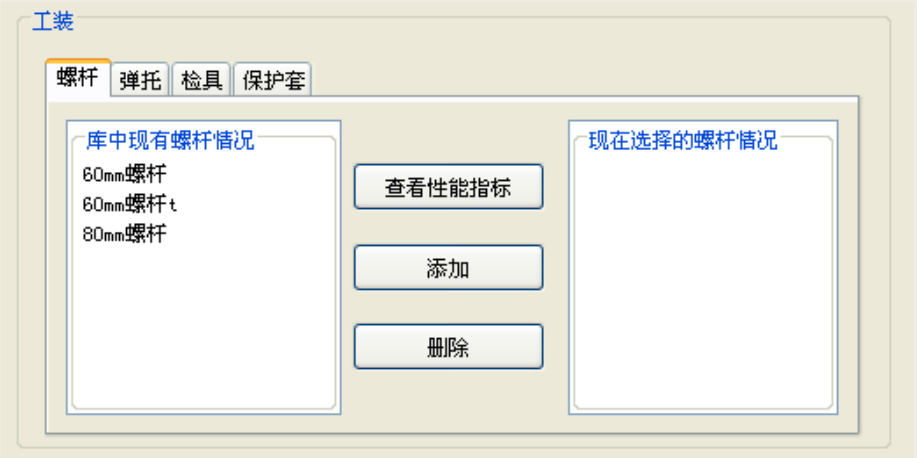
用户界面测试-符合标准和规范

- 通常标准是已经确立的，多数用户已经熟悉并接受了这些标准和规范、或已经认同了这些信息所代表的意义
- 如果软件在某一个平台上运行，就需要把该平台的标准和规范作为产品规格说明书的补充内容，在建立测试案例时和产品规格说明书一样作为依据

用户界面测试-一致性

- 与用户的现实世界的一致性
- 与用户常用或习惯的软件产品的一致性
- 不同用户界面之间的一致性





用户界面测试-直观性

- 首先了解所需的功能或期待的响应应该明显，并在预期的地方出现。
- 其次要考虑用户界面的组织和布局是否合理

用户界面测试-灵活性

- 不同用户所使用的功能和数据存在差异，这种差异应反映到界面，即要为不同用户提供合适的个性化界面或选择的灵活性

用户界面测试

- 舒适性
  - *尽可能降低用户操作的复杂性，尽量减少结构层次及用户操作量*
  - *用户界面的外观、风格与用户的工作性质和环境协调*
  - *错误处理*
- 正确性
  - *测试是否做了该做的事*
- 宽容性
  - *用户错误操作的宽容*

性能测试

- 性能测试用来测试软件在系统集成中的运行性能，特别是针对实时系统和嵌入式系统，仅提供符合功能需求但不符合性能需求的软件是不能被接受的。
- 性能测试可以在测试过程的任意阶段进行，但只有当整个系统的所有成分都集成在一起后，才能检查一个系统的真正性能。
- 性能测试常常和强度（压力）测试结合起来进行，而且常常需要硬件和软件测试设备，这就是说，常常有必要在一种苛刻的环境中衡量资源的使用（比如，处理器周期）
- 性能测试(Performance test)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、占有系统资源等方面的系统数据

性能测试的目的和需求

- 目的：
  - *为了验证系统是否达到用户提出的性能指标，同时发现系统中存在的性能瓶颈，起到优化系统的目的。*
- 性能测试需求：
  - *用户对各项指标提出的明确需求；如果用户没有提出性能指标则根据用户需求、测试设计人员的经验来设计各项测试指标。（需求+经验）*
- 主要的性能指标：
  - *服务器的各项指标（CPU、内存占用率等）、后台数据库的各项指标、网络流量、响应时间*

性能测试要点

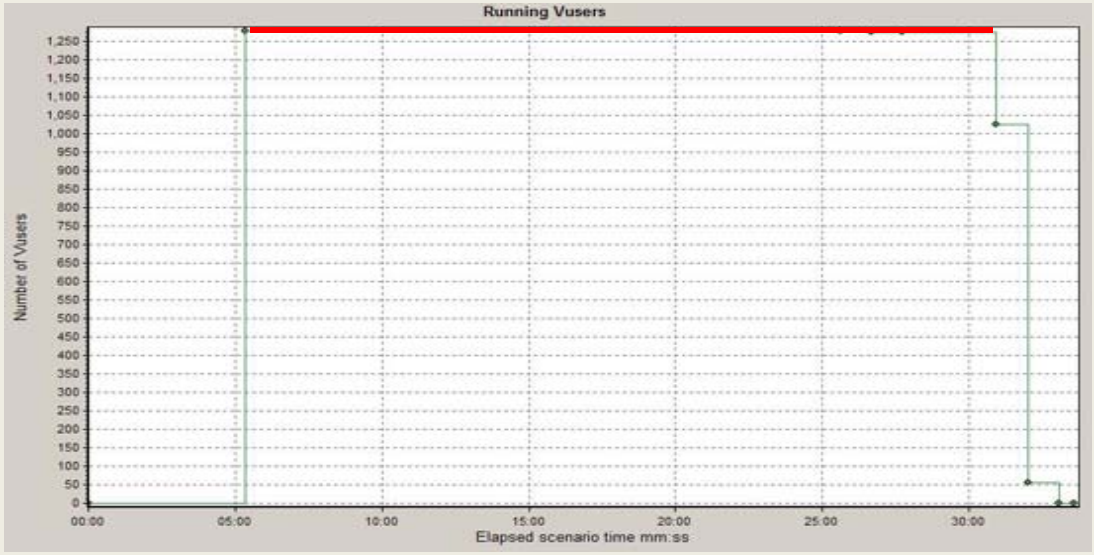
- 性能测试包括以下几个方面
  - *评估系统的能力：测试中得到的负荷和响应时间等数据可以被用于验证所计划的模型的能力，并帮助做出决策。*
  - *识别系统中的弱点：受控的负荷可以被增加到一个极端的水平并突破它，从而修复系统的瓶颈或薄弱的地方。*
  - *系统调优：重复运行测试，验证调整系统的活动得到了预期的结果，从而改进性能，检测软件中的问题*
- 测试环境应尽量与产品运行环境保持一致，应单独运行尽量避免与其他软件同时使用
- 性能测试一般使用测试工具和测试人员编制测试脚本来完成
- 性能测试的重点在于前期数据的设计与后期数据的分析
- 性能测试的用例主要涉及到整个系统架构的问题，所以测试用例一旦生成，改动一般不大，所以做性能测试的重复使用率一般比较高

性能测试方法

- 负载模拟：
  - *并发用户 + 思考时间 + 每次请求的数据量+ 负载模式。*
- 性能测试步骤：
  - *确定性能测试需求*
  - *根据测试需求，选择测试工具和开发相应的测试脚本*
  - *建立性能测试负载模型，就是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的时间等*
  - *执行性能测试*
  - *结果分析，并提交性能测试报告*
- 两种负载类型
  - *“flat”测试*
  - *ramp-up测试*
- 对于企业级的系统，性能测试的方法主要有：
  - *基准测试*
  - *性能规划测试*
  - *渗入测试*
  - *峰谷测试*

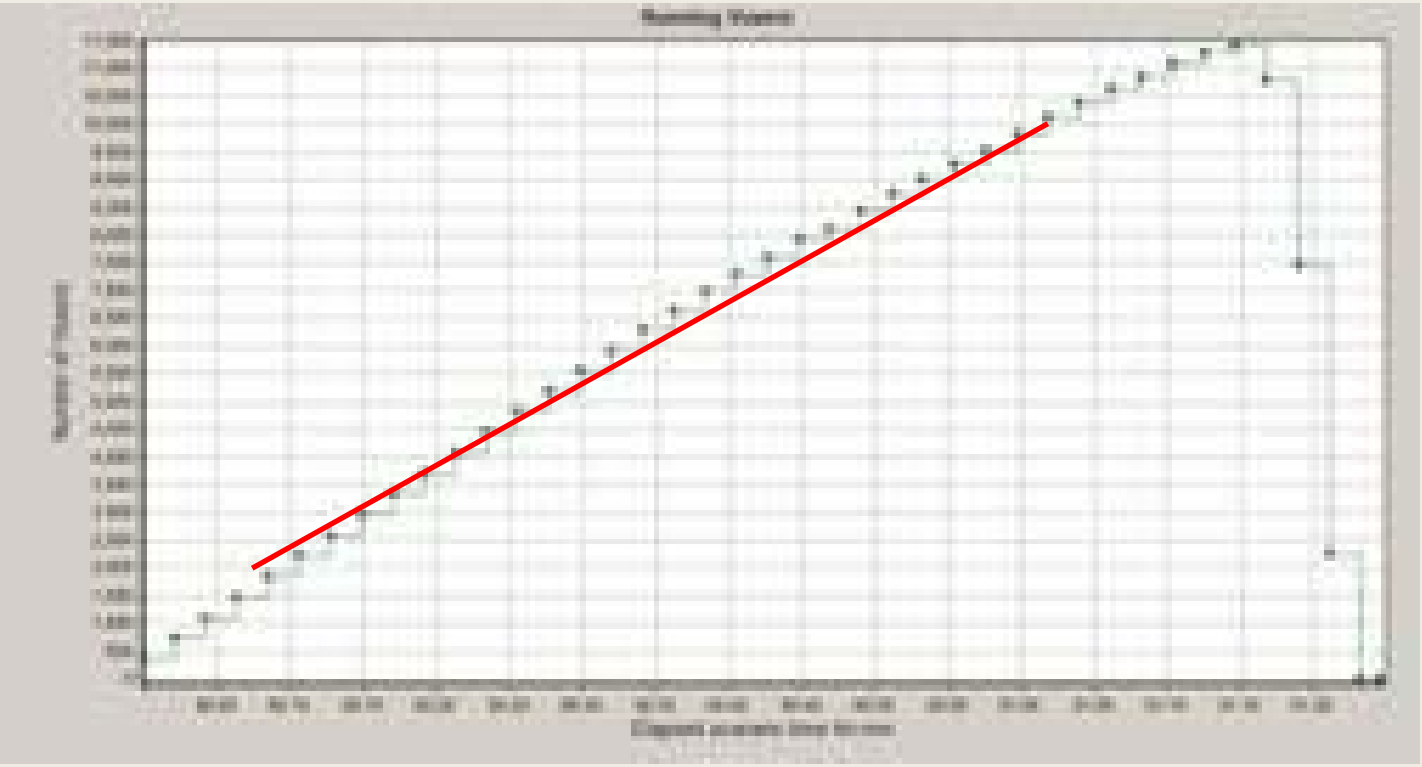
性能测试-Flat测试

- 对于一次给定的测试，应该取响应时间和吞吐量的平均值。精确地获得这些值的唯一方法是一次加载所有的用户，然后在预定的时间段内持续运行



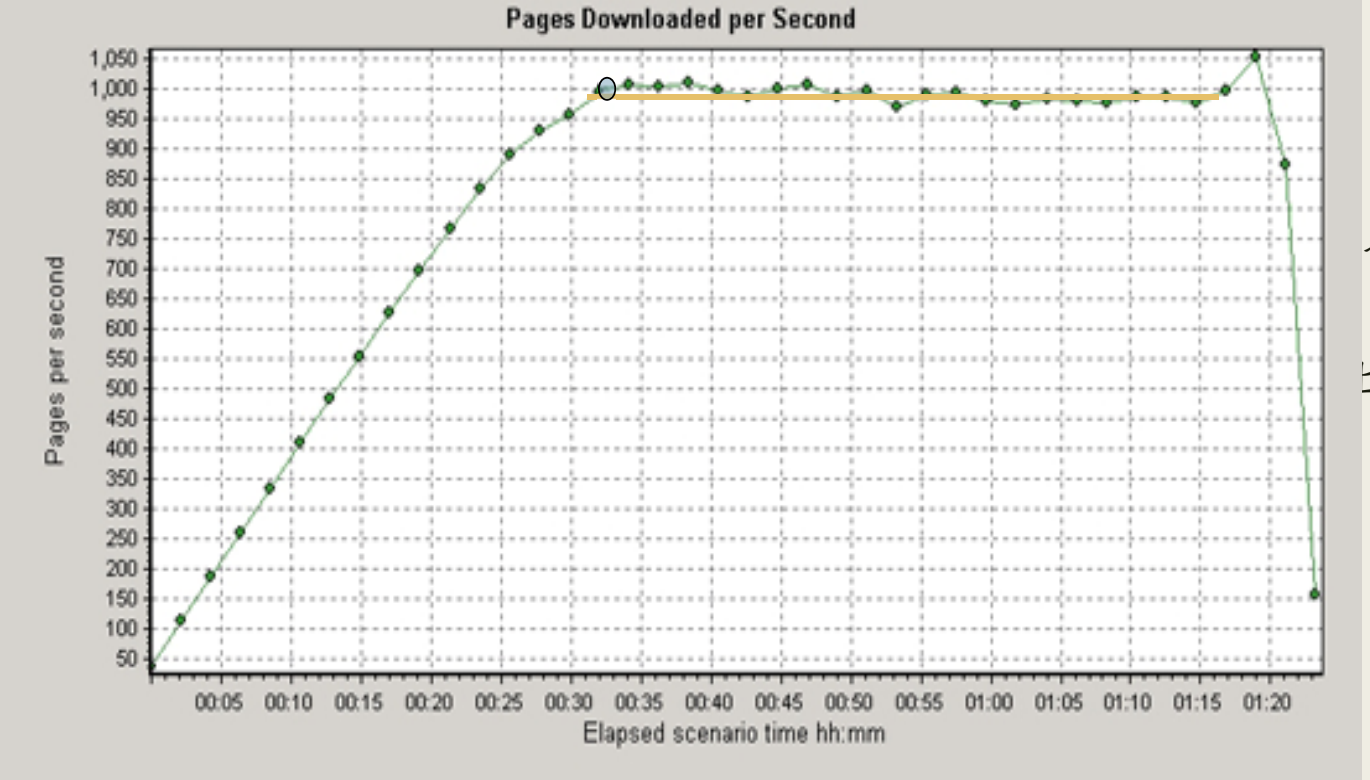
性能测试- Ramp-up测试

- 用户是交错上升的（每几秒增加一些新用户）。ramp-up测试不能产生精确和可重现的平均值，这是因为由于用户的增加是每次一部分，系统的负载在不断地变化。其优点是，可以看出随着系统负载的改变，测量值是如何改变的→据此选择要运行的flat测试的范围。



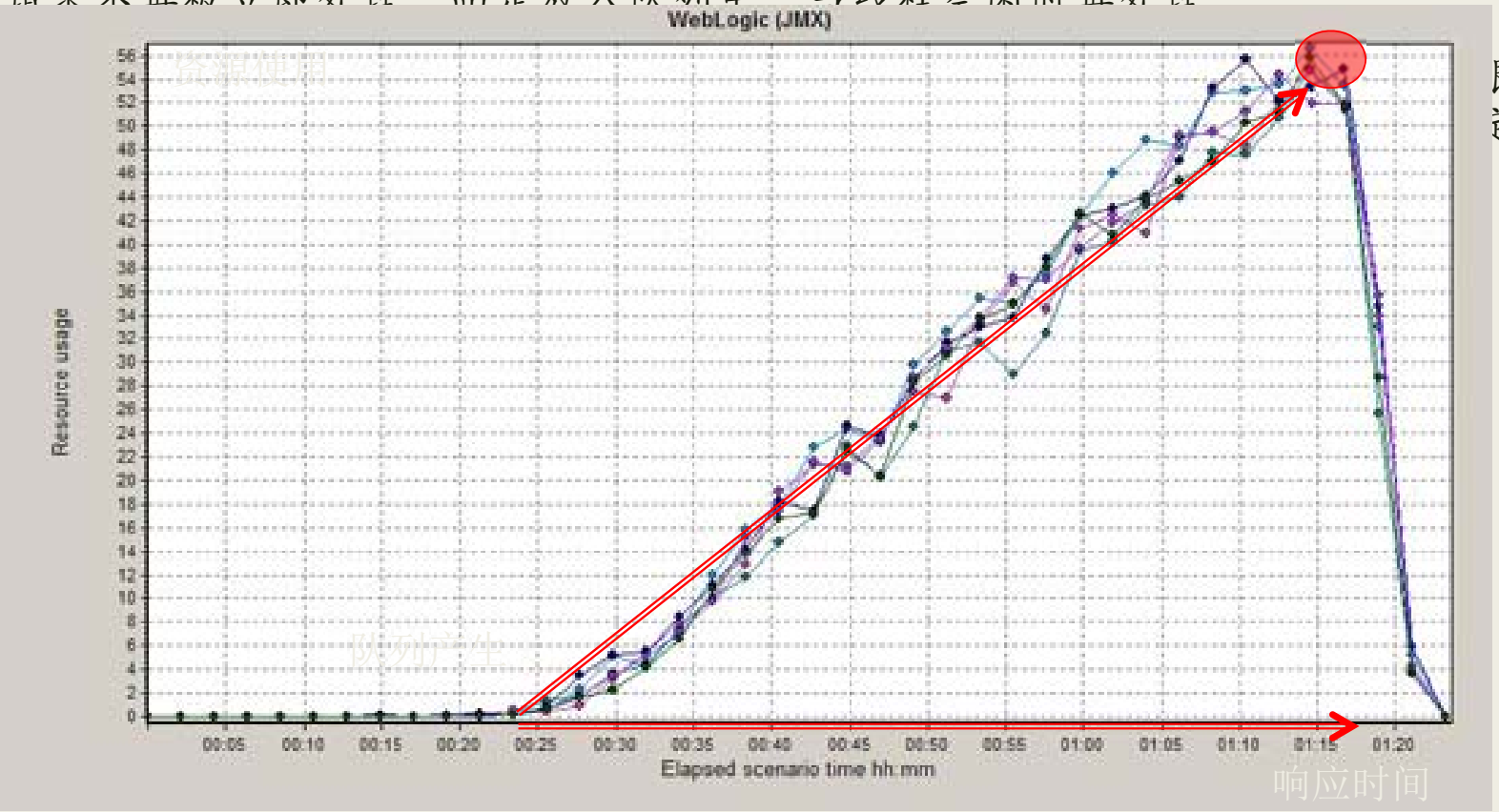
性能测试- 基准测试

- 基准测试的关键是要获得一致的、可再现的结果。
- 假定测试的两个指标是服务器的响应时间和吞吐量，会受到负载的影响。而负载又受两个因
  - *同时与*
  - *每个虚*
- 与服务器通 也越大。这两
- 随着服务器 上稳定下来



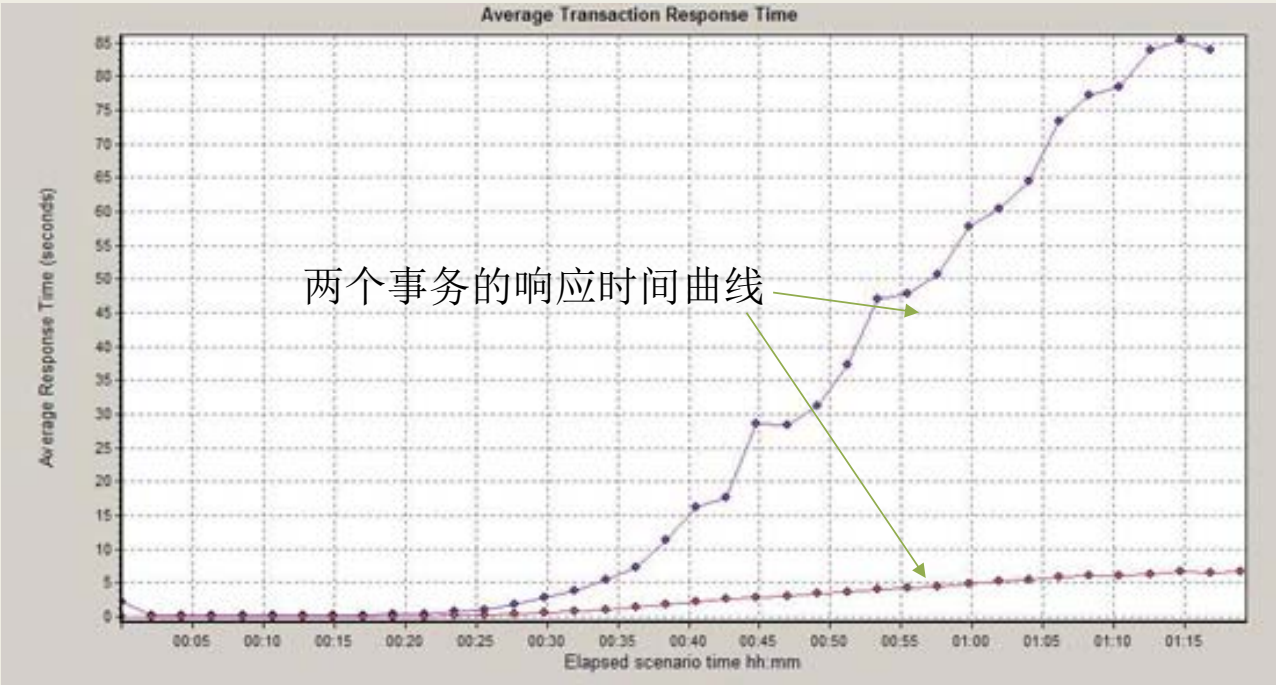
性能测试- 基准测试（2）

- 在某一点上，执行队列开始增长，因为服务器上所有的线程都已投入使用，传入的请求不再被立即处理，而是放入队列中，当线程空闲时再处理
- 当 很。
  但 定



性能测试- 基准测试（3）

- 将系统置于相同的高负载下，将请求之间间隔时间设为零。这样服务器会立即超载，并开始构建执行队列。如果请求（虚拟用户）数保持一致，基准测试的结果会非常精确→ flat运行是获得基准测试数据的理想模式



性能测试- 基准测试的思考

- 对用户来讲，响应时间的长短并没有绝对的区别
  - *例如一个税务报账系统，用户每月使用一次该系统，每次进行数据录入等操作需要2小时以上的时间，当用户选择提交后，即使系统在20分钟后才给出处理成功的消息，用户仍然不会认为系统的响应时间不能接受。*
  - *因为相对于一个月才进行一次的操作来说，20分钟是一个可以接受的等待时间。所以在进行性能测试的时候，合理的响应时间取决于实际的用户需求，而不能根据测试人员自己的设想来决定*
- 在实际的性能测试中，经常接触到的与并发用户数相关的概念还包括“系统用户数”、“同时在线用户人数”和“同时操作用户数”。
  - *例如，某大学的网站有邮件服务及学生信息、选课系统等业务，基于这些业务的用户共有10000人，则这10000个用户就称为系统用户数。假设整个网站在最高峰时有6000人同时在线，则这6000人可以称作同时在线用户人数。假设600人同时请求操作，则600人作为系统的并发用户数*

性能测试-性能规划测试

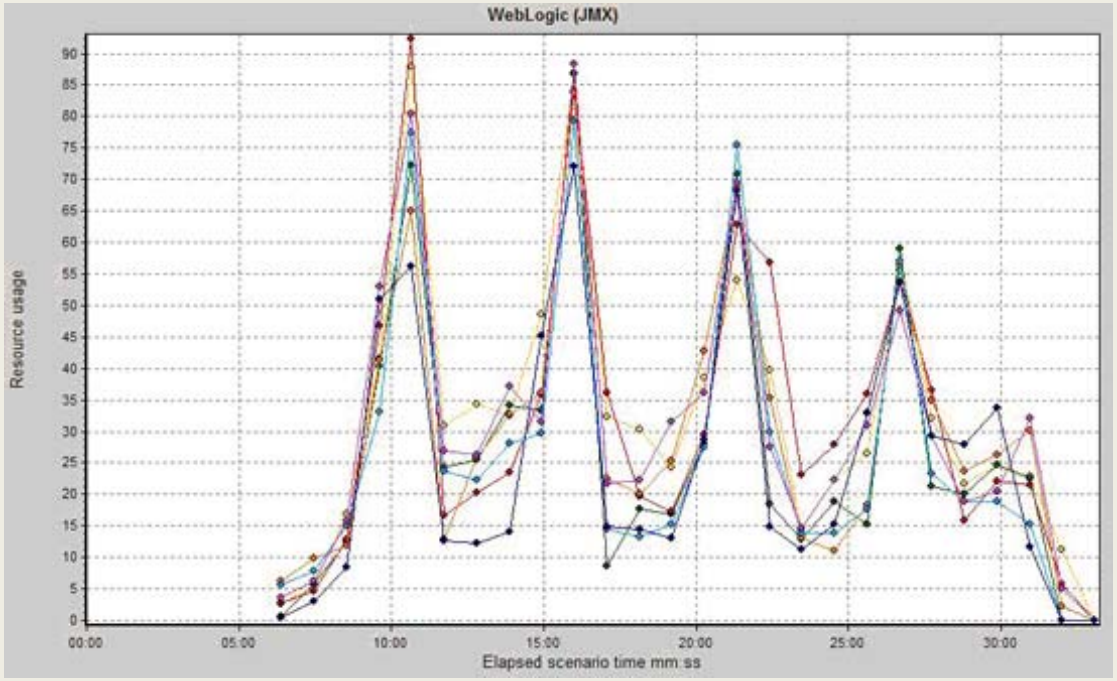
- 性能规划测试的目标是找出在特定的环境下，给定应用程序的性能可以达到何种程度。例如，如果要以5秒或更少的响应时间支持8,000个当前用户，需要多少个服务器?
- 要确定系统的容量，需要考虑几个因素:
  - *用户中有多少是并发与服务器通信的。*
  - *每个用户的请求间时间间隔是多少。*
- 如何加载用户以模拟负载状态?
  - *最好的方法是模拟高峰时间用户与服务器通信的状况。*
  - *如果用户负载状态是在一段时间内逐步达到的,选择ramp-up测试，每隔几秒增加x个用户;*
  - *如果所有用户是在一个非常短的时间内同时与系统通信，就应该使用flat测试，将所有的用户同时加载到服务器.*
- 什么是确定容量的最好方法?
  - *结合两种负载类型的优点，并运行一系列的测试;如：首先使用ramp-up测试确定系统支持的用户范围 该范围内不同的并发用户负载进行一系列的flat测试，更精确地确定系统的容量。*

性能测试-渗入测试

- 渗入测试是一种比较简单的性能测试。渗入测试所需时间较长，它使用固定数目的并发用户测试系统的总体健壮性。这些测试将会通过内存泄漏、增加的垃圾收集(GC)或系统的其他问题，显示因长时间运行而出现的任何性能降低。
- 建议运行两次测试——一次使用较低的用户负载（要在系统容量之下，以便不会出现执行队列），一次使用较高的负载（以便出现积极的执行队列）。

性能测试-峰谷测试

- 兼有容量规划ramp-up测试和渗入测试的特征,目标是确定从高负载（例如系统高峰时间的负载）恢复、转为几乎空闲、然后再攀升到高负载、再降低的能力。



强度测试

- 强度测试（也称压力测试-Stree Testing）的目的是要检测非正常的情形，测试是想要破坏程序。
- 通过模拟实际应用的软硬件环境及用户使用过程的系统负荷，逐渐加载或一次性加载，长时间或超大负荷地运行软件，以测试系统的稳定性，并试图找出系统性能的瓶颈和异常的地方
- 强度测试需要在反常规数据量、频率或资源的方式下运行系统，以检验系统能力的最高实际限度。
- 举例：
  - *如果正常的中断频率为每秒5次，强度测试设计为每秒50次中断。*
  - *把输入数据的量提高一个数量级来测试输入功能会如何响应。*
  - *若某系统正常运行可支持200个终端并行工作，强度测试则检验1000个终端并行工作的情况。*
  - *运行大量的消耗内存或其他系统资源的测试实例。*

强度测试

- 与性能测试的联系与区别
  - *强度测试用来保证产品发布后系统能否满足用户需求，关注的重点是系统整体；性能测试可以发生在各个测试阶段，即使是在单元层，一个单独模块的性能也可以进行评估。*
  - *强度测试是通过确定一个系统的瓶颈，来获得系统能提供的最大服务级别的测试。性能测试是检测系统在一定负荷下的表现，是正常能力的表现；而压力测试是极端情况下的系统能力的表现。*
    - 例如对一个网站进行测试，模拟10到50个用户同时在线并观测系统表现，是常规性能测试；当用户增加到系统出项瓶颈时，如1000乃至上万个用户时，则为强度测试

强度测试方法

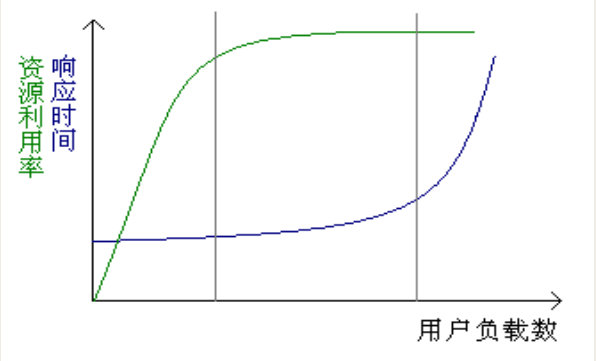
- 重复测试
  - *重复测试就是一遍又一遍地执行某个操作或功能，比如重复调用一个Web服务。*
  - *强度测试的一项任务就是确定在极端情况下一个操作能否正常执行，并且能否持续不断地在每次执行时都正常。这对于推断一个产品是否适用于某种生产情况至关重要，客户通常会重复使用产品。重复测试往往与其它测试手段一并使用*
- 并发测试
  - *并发是同时执行多个操作的行为，即在同一时间执行多个测试线程。*
  - *例如，在同一个服务器上同时调用许多Web服务。并发测试原则上不一定适用于所有产品，但多数软件都具有某个并发行为或多线程行为元素，这一点只能通过执行多个代码测试用例才能得到测试结果*

强度测试方法（续）

- 量级增加
  - *压力测试可以重复执行一个操作，但是操作自身也要尽量给产品增加负担。*
    - 例如一个Web服务允许客户机输入一条消息，测试人员可以通过模拟输入超长消息来使操作进行高强度的使用，即增加这个操作的量级。这个量级的确定总是与应用系统有关，可以通过查找产品的可配置参数来确定量级。例如，数据量的大小、延迟时间的长度、输入速度以及输入的变化等
- 随机变化
  - *该手段是指对上述测试手段进行随机组合，以便获得最佳的测试效果。*
    - 例如，使用重复时，在重新启动或重新连接服务之前，可以改变重复操作间的时间间隔、重复的次数，或者也可以改变被重复的Web服务的顺序。
    - 使用并发时，可以改变一起执行的Web服务、同一时间运行的Web服务数目，也可以改变关于是运行许多不同的服务还是运行许多同样的实例的决定。
    - 量级测试时，每次重复测试时都可以更改应用程序中出现的变量（例如发送各种大小的消息或数字输入值）

容量测试

- 采用特定的手段测试系统能够承载处理任务的极限值所从事的测试工作
- 目的：使系统承受超额的数据容量来发现它是否能够正确处理
- 容量测试的任务
  - *确定被测系统数据量的极限*
- 容量测试条件
  - *链接或模拟了最大（实际或实际允许）数量的客户机。*
  - *所有客户机在长时间内执行相同的、可能性能不稳定的重要业务功能。*
  - *已达到最大的数据库大小（实际的或按比例缩放的），而一起同时执行多个查询或报表事务*



容量测试步骤

- 分析系统的外部数据源，并进行分类
- 对每类数据源分析可能的容量限制，对于记录类型数据需要分析记录长度限制，记录中每个域长度限制和记录数量限制。
- 对每个类型数据源，构造大容量数据对系统进行测试。
- 分析测试结果，并与期望值比较，确定目前系统的容量瓶颈。
- 对系统进行优化并重复以上四步，直到系统达到期望的容量处理能力。
- 对系统进行优化并重复以上四步，直到系统达到期望的容量处理能力

常见的容量测试例子

- 处理数据敏感操作时进行的相关数据比较
- 使用编译器编译一个极其庞大的源程序
- 使用一个链接编译器编辑一个包含成千上万模块的程序
- 一个电路模拟器模拟包含成千上万块的电路
- 一个操作系统的任务队列被充满
- 一个测试形式的系统被灌输了大量文档格式
- 互联网中庞大的E-mail信息和文件信息

恢复测试

- 恢复测试是通过各种手段，强制性地使软件出错，使其不能正常工作，进而检验系统的恢复能力。
- 恢复测试包含的内容：
  - *如果系统恢复是自动的（由系统自身完成），则应该检验：重新初始化、检验点设置机构、数据恢复以及重新启动是否正确。*
  - *如果这一恢复需要人为干预，则应考虑平均修复时间是否在限定的、可以接受的范围之内*
- 恢复测试中需要检查以下各项：
  - *错误探测功能*
  - *能否切换或启动备用的硬件；*
  - *在故障发生时能否保护正在运行的作业和系统状态；*
  - *在系统恢复后能否从最后记录下来的无错误状态开始继续执行作业，等等。*
  - *掉电测试*

安全测试

- 安全测试的目的在于验证安装在系统内的保护机制能否在实际中保护系统且不受非法入侵，不受各种非法干扰。
- 在安全测试中，测试者扮演着试图攻击系统的个人角色：
  - *尝试去通过外部的手段来获取系统的密码*
  - *使用可以瓦解任何防守的客户软件来攻击系统*
  - *把系统“瘫痪”，使得其他用户无法访问*
  - *有目的地引发系统错误，期望在恢复过程中侵入系统*
  - *通过浏览非保密的数据，从中找到进入系统的钥匙*
- 系统的安全测试要设置一些测试用例试图突破系统的安全保密措施，检验系统是否有安全保密的漏洞
- 对软件产品安全测试应侧重于以下方面：用户对数据或业务功能的访问控制，数据存储和数据通信的远程安全控制。
  - *用户管理和访问控制*
  - *通信加密*
  - *安全日志测试*

安全测试

- 系统安全设计的准则
  - 使非法侵入的代价超过被保护的信息的价值，从而令非法侵入者无利可图。
  - 一般来讲，如果黑客为非法入侵花费的代价（考虑时间、费用、危险等因素）高于得到的好处，那么这样的系统可以认为是安全的系统
- 测试者扮演一个试图攻击系统的角色
  - 尝试通过外部的手段来获取系统的密码
  - 使用能够瓦解任何防护的客户软件来攻击系统
  - 把系统“制服”，使别人无法访问
  - 有目的地引发系统错误，期望在系统恢复过程中侵入系统
  - 通过浏览非保密的数据，从中找到进入系统的钥匙等

- 安全性一般分为两个层次
  - *应用程序级别的安全性：对数据或业务功能的访问*
  - *系统级别的安全性：对系统的登录或远程访问*
- 二者的关系
  - *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数据。*
    - 例如，某财务系统可能会允许所有人输入数据，创建新账户，但只有管理员才能删除这些数据或账户。
  - *系统级别的安全性对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入口来访问*

安全性测试方法-功能验证

- 功能验证：采用软件测试当中的黑盒测试方法，对涉及安全的软件功能，如用户管理模块、权限管理模块、加密系统、认证系统等进行测试，主要是验证上述功能是否有效
- 功能性的安全性问题包括
  - *控制特性是否工作正确?*
  - *无效的或者不可能的参数是否被检测并且适当的处理?*
  - *无效的或者超出范围的指令是否被检测并且适当的处理?*
  - *错误和文件访问是否适当的被记录?*
  - *是否有变更安全性表格的过程?*
  - *系统配置数据是否能正确保存，系统故障时是否能恢复?*
  - *系统配置数据能否导出，在其他机器上进行备份?*
  - *系统配置数据能否导入，导入后能否正常使用?*
  - *系统配置数据保存时是否加密?*
  - *没有口令是否可以登录到系统中?*
  - *有效的口令是否被接受，无效的口令是否被拒绝?*

安全性测试方法-功能验证

- 功能验证：采用软件测试当中的黑盒测试方法，对涉及安全的软件功能，如用户管理模块、权限管理模块、加密系统、认证系统等进行测试，主要是验证上述功能是否有效
- 功能性的安全性问题包括
  - *系统对多次无效口令是否有适当的反应?*
  - *系统初始的权限功能是否正确?*
  - *各级用户权限划分是否合理?*
  - *用户的生命期是否有限制?*
  - *低级别的用户是否可以操作高级别用户命令?*
  - *高级别的用户是否可以操作低级别用户命令?*
  - *用户是否会自动超时退出，超时的时间是否设置合理，用户数据是否会丢失?*
  - *登录用户修改其他用户的参数是否会立即生效?*
  - *系统在最大用户数量时是否操作正常?*
  - *对于远端操作是否有安全方面的特性?*
  - *防火墙是否能被激活和取消激活?*
  - *防火墙功能激活后是否会引起其他问题?*

安全性测试方法-漏洞扫描

- 安全漏洞扫描通常都是借助于特定的漏洞扫描器完成
  - *漏洞扫描器是一种能自动检测远程或本地主机安全性弱点的程序，通过使用漏洞扫描器，系统管理员能够发现所维护信息系统存在的安全漏洞，从而在信息系统网络安全防护过程中做到有的放矢，及时修补漏洞。*
- 安全漏洞扫描是可以用于日常安全防护，同时可以作为对软件产品或信息系统进行测试的手段，可以在安全漏洞造成严重危害前发现漏洞并加以防范
- 按常规标准，可以将漏洞扫描器分为两种类型：主机漏洞扫描器（Host Scanner）和网络漏洞扫描器（Network Scanner）
  - *主机漏洞扫描器是指在系统本地运行检测系统漏洞的程序，如著名的COPS、Tripewire、Tiger等自由软件。*
  - *网络漏洞扫描器是指基于网络远程检测目标网络和主机系统漏洞的程序，如Satan、ISS Internet Scanner等。*

安全性测试方法-模拟攻击

- 模拟攻击试验是一组特殊的黑盒测试案例，通常以模拟攻击来验证软件或信息系统的安全防护能力。
  - *冒充*
  - *重演*
  - *消息篡改*
  - *拒绝服务*
  - *内部攻击*
  - *外部攻击*
  - *陷阱*
  - *木马*

安全性测试方法-模拟攻击

- 口令猜测
  - *一旦黑客识别了一台主机，而且发现了基于NetBIOS、Telnet或NFS服务的可利用的用户账号，并成功地猜测出了口令，就能对机器进行控制。*
- 缓冲区溢出
  - *在服务程序中，如果程序员使用类似于strcpy()、strcat ()等字符串函数，由于这些函数不进行有效位检查，所以可能导致恶意用户编写一小段程序来进一步打开缺口，将该代码放在缓冲区有效载荷末尾。这样，当发生缓冲区溢出时，返回指针指向恶意代码，执行恶意指令，就可以得到系统的控制权*
- 当一个消息或部分消息为了产生非授权效果而被重复时，就出现了重演
  - *例如，一个含有鉴别信息的有效消息可能被另一个实体所重演，目的是鉴别它自己。*
- DNS高速缓存污染
  - *由于DNS服务器与其他名称服务器交换信息的时候并不进行身份验证，这就使黑客可以加入不正确的信息，并把用户引向黑客自己的主机。*
- 伪造电子邮件
  - *由于SMTP并不对邮件发送附件的身份进行鉴定，因此黑客可以对内部客户伪造电子邮件，声称是来自某个客户认识并相信的人，并附上可安装的特洛伊木马程序，或者是一个指向恶意网站的链接。*

安全性测试方法-模拟攻击

- 服务拒绝
  - *当一个实体不能执行它的正常功能，或它的动作妨碍了别的实体执行它们的正常功能的时候，便发生服务拒绝。*
    - 可能是一般性的，比如一个实体抑制所有的消息
    - 也可能是有具体目标的，例如一个实体抑制所有流向某一特定目的端的消息，如安全审计服务
    - 死亡之Ping、泪滴、UDP洪水、SYN洪水、Land攻击、Smurf攻击、Fraggle攻击、电子邮件炸弹、畸形消息攻击
- 内部攻击
  - *当系统的合法用户以非故意或非授权方式进行动作时就成为内部攻击。*
  - *防止内部攻击的保护方法*
    - 所有管理数据流进行加密
    - 利用包括使用强口令在内的多级控制机制和集中管理机制来加强系统的控制能力
    - 为分布在不同场所的业务部门划分VLAN，将数据流隔离在特定部门
    - 利用防火墙为进出网络的用户提供认证功能，提供访问控制保护
    - 使用安全日志记录网络管理数据流等

安全性测试方法-模拟攻击

- 外部攻击
  - *搭线窃听（主动的与被动的）*
  - *截取辐射*
  - *冒充为系统的授权用户*
  - *冒充为系统的组成部分*
  - *为鉴别或访问控制机制设置旁路等*
- 陷阱
  - *当系统的实体受到改变，致使一个攻击者能对命令或对预定的事件或事件序列产生非授权的影响时，其结果就称为陷阱门。*
    - 例如：口令的有效性可能被修改，使其除了正常效力之外也使攻击者的口令生效
- 木马
  - *对系统而言的特洛伊木马，是指它不但具有自己的授权功能，而且还有非授权功能。*
    - 一个向非授权信道拷贝消息的中继就是一个特洛伊木马
    - 典型的特洛伊木马有NetBus、BackOffice和BO2k等

安全性测试方法-侦听技术

- 侦听技术实际上是在数据通信或数据交互过程，对数据进行截取分析的过程。
- 目前最为流行的是网络数据包的捕获技术，通常称为Capture，黑客可以利用该项技术实现数据的盗用，而测试人员同样可利用该项技术实现安全测试。
- 该项技术主要用于对网络加密的验证。

配置测试

- 目标：在不同的硬件配置下，检查系统是否发生功能或者性能上的问题
- 方法：一般需要建立测试实验室

- 计划配置测试时一般采用的过程
  - *确定所需的硬件类型*
  - *确定哪些硬件型号和驱动程序可以使用*
  - *确定可能的硬件特性、模式和选项*
  - *将硬件配置缩减到可以控制的范围内*
  - *明确使用硬件配置的软件的特性*
  - *设计在每种配置中执行的测试用例*
  - *反复测试直到对结果满意为止*

正确性测试

- 正确性测试检查软件的功能是否符合规格说明。
- 正确性测试的方法：
  - *枚举法，即构造一些合理输入，检查是否得到期望的输出。测试时应尽量设法减少枚举的次数，关键在于寻找等价区间，因为在等价区间中，只需用任意值测试一次即可。*
  - *边界值测试，即采用定义域或者等价区间的边界值进行测试。因为程序设计容易疏忽边界情况，程序也容易在边界值处出错。*

可靠性测试

- 可靠性测试是从验证的角度出发，检验系统的可靠性是否达到预期的目标，同时给出当前系统可能的可靠性增长情况。
- 对可靠性性测试来说，最关键的测试数据包括失效间隔时间，失效修复时间，失效数量，失效级别等。根据获得的测试数据，应用可靠性模型，可以得到系统的失效率及可靠性增长趋势。
- 可靠性指标有时很难确定，通常采用平均无故障时间或系统投入运行后出现的故障不能大于多少数量这些指标来对可靠性进行评估
- 通常使用以下几个指标来度量系统的可靠性：
  - *平均失效等待时间 （MTTF，Mean Time To Failure）*
  - *平均失效间隔时间（MTBF，Mean Time Between Failure）*
  - *因故障而停机的时间在一年中应不超过多少时间*

兼容性测试

- 软件兼容性测试是检测各软件之间能否正确地交互和共享信息，其目标是保证软件按照用户期望的方式进行交互，使用其它软件检查软件操作的过程。
- 兼容性的测试通常需要解决以下问题：
  - *新开发的软件需要与哪种操作系统、Web浏览器和应用软件保持兼容，如果要测试的软件是一个平台，那么要求应用程序能在其上运行。*
  - *应该遵守哪种定义软件之间交互的标准或者规范。*
  - *软件使用何种数据与其它平台、与新的软件进行交互和共享信息*

兼容性测试-示例

表：兼容性测试矩阵示例

应用程序	操作系统					
	Windows 98	Windows 98 SE	Windows Me	Windows NT 4.0	Windows 2000	Windows XP
IE 5.5	*					
IE 6.0	*					
IE 7.0	*	*	*	*	***	***
Media Player	*	*	*			***
RealPlayer	*	***	*			***

Web网站测试

- Web网站的网页是由文字、图形、音频、视频和超级链接组成的文档。
- 对网站的测试包含许多方面，如配置测试、兼容测试、可用性测试、文档测试等；黑盒测试、白盒测试、静态测试和动态测试都有可能采用。
- 通常Web网站测试包含以下内容：
  - *文字测试*
  - *链接测试*
  - *图像、图像测试*
  - *表单测试*
  - *动态内容测试*
  - *数据库测试*
  - *服务器性能及负载测试*
  - *安全性测试*

文档测试-文档

- 软件产品由可运行的程序、数据和文档组成。
- 文档是软件的一个重要组成部分。在软件的整个生命周期中，会产生许多文档，在各个阶段中以文档作为前阶段工作成果的总结和后阶段工作的依据。
- 一个好的软件文档能从以下3方面提高软件产品的整体质量。
  - *提高可用性：可用性大都与软件文档有关*
  - *提高可靠性：可靠性是指软件平稳运行的程度*
  - *降低支持费用：好的文档能够通过恰当的解释和引导帮助用户克服困难，尽可能预防这种情况发生*
- 文档测试：针对系统提交给用户的文档进行验证，目标是验证软件文档是否正确记录系统的开发全过程的技术细节。通过文档测试可以改进系统的可用性、可靠性、可维护性和安装性
- 软件文档的分类如下表所示：

用户文档	开发文档	管理文档
用户手册、操作手册、维护修改建议	软件需求说明书、数据库设计说明书、概要设计说明书、详细设计说明书、可行性研究报告	项目开发计划、测试计划、测试报告、开发进度月报、开发总结报告

用户文档测试的内容

- 用户文档测试的内容
- *把用户文档作为测试用例选择依据*
- *确切的按照文档所描述的方法使用系统*
- *测试每个提示和建议，检查每条陈述*
- *查找容易误导用户的内容*
- *把缺陷并入缺陷跟踪库*
- *测试每个在线帮助超链接*
- *测试每条语句，不要想当然*
- *表现的像一个技术编辑而不是一个被动的评审者*
- *首先对整个文档进行一般的评审，然后进行一个详细的评审*
- *检查所有的错误信息*
- *测试文档中提供的每个样例*
- *保证所有索引的入口有文档文本*
- *保证文档覆盖所有关键用户功能*
- *保证阅读类型不是太技术化*
- *寻找相对比较弱的区域，这些区域需要更多的解释*

开发文档测试的内容

- 系统定义的目标是否与用户的要求一致
- 系统需求分析阶段提供的文档资料是否齐全
- 文档中的所有描述是否完整、清晰，准确地反映用户要求
- 与所有其他系统成份的重要接口是否都已经描述
- 被开发项目的数据流与数据结构是否足够、确定
- 所有图表是否清楚，在不补充说明时能否理解
- 主要功能是否已包括在规定的软件范围之内，是否都已充分说明
- 软件的行为和它必须处理的信息、必须完成的功能是否一致
- 设计的约束条件或限制条件是否符合实际
- 是否考虑了开发的技术风险
- 是否考虑过软件需求的其他方案
- 是否考虑过将来可能会提出的软件需求
- 是否详细制定了检验标准，它们能否对系统定义是否成功进行确认
- 有没有遗漏、重复或不一致的地方
- 用户是否审查了初步的用户手册或原型
- 项目开发计划中的估算是否受到了影响

开发文档测试包括

- 接口
  - *分析软件各部分之间的联系，确认软件的内部接口与外部接口是否已经明确定义。模块是否满足高内聚低耦合的要求。模块作用范围是否在其控制范围之内*
- 风险
  - *确认该软件设计在现有的技术条件下和预算范围内是否能按时实现*
- 实用性
  - *确认该软件设计对于需求的解决方案是否实用*
- 技术清晰度
  - *确认该软件设计是否以一种易于翻译成代码的形式表达*
- 可维护性
  - *从软件维护角度出发，确认该软件设计是否考虑了方便未来的维护*
- 质量
  - *确认该软件设计是否表现出良好的质量特征*
- 各种选择方案
  - *看是否考虑过其他方案，比较各种选择方案的标准是什么*
- 限制
  - *评估对该软件的限制是否实现，是否与需求一致*
- 其他具体问题
  - *对于文档、可测试性、设计过程等进行评估*

文档测试方法

- *文档走查*
  - 熟悉软件特性的人，通过阅读文档，来检查文档的质量
- *数据校对*
  - 只需检查文档中数据所在部分，不必检查全部文档。数据有：边界值、程序的版本、硬件配置、参数缺省值等
- *操作流程检查*
  - 安装/卸载过程、参数配置过程、功能操作和向导功能
- *引用测试*
  - 文档之间的相互引用，如术语、图、表和示例等，是Bug的多发处

	语言类错误	版面类错误	逻辑类错误	一致性错误	联机文档功能错误
文档走查	√	√	√	√	√
数据校对			√	√	
操作流程检查			√		√
引用测试				√	

系统测试工具

- LoadRunner
- Ttworkbench
- QACenter
- DataFactory
- JMeter

验收测试

概述

- 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
- 验收测试完成的准则
  - *规定的所有验收测试用例已经运行*
  - *对软件缺陷的所有修改都已进行了回归测试*
  - *发现的缺陷已经解决或关闭*
  - *达到预期的覆盖率目标，一般要求产品特性或业务需求覆盖率为100%*
  - *修改软件缺陷后，所有相关的软件文档的版本均已经更新*
  - *获得用户/客户签署的验收测试报告*
- 验收测试完成后的主要交付物
  - *验收测试计划*
  - *验收测试用例及测试数据*
  - *软件缺陷报告*
  - *验收测试报告*
  - *通过验收测试的，用于正式运行的系统版本*
  - *通过验收测试的代码及更新后的相关软件文档*

验收测试的首要条件

- 软件开发已经完成，并全部解决了已知的软件缺陷；
- 验收测试计划已经过评审并批准，并且置于文档控制之下；
- 对软件需求说明书的审查已经完成；
- 对概要设计、详细设计的审查已经完成；
- 对所有关键模块的代码审查已经完成；
- 对单元、集成、系统测试计划和报告的审查已经完成；
- 所有的测试脚本已完成，并至少执行过一次，且通过评审；
- 使用配置管理工具且代码置于配置控制之下；
- 软件问题处理流程已经就绪；
- 新系统已通过尝试运行工作；
- 所被测的新系统应该是稳定的，要符合技术文档和标准的规定；
- 已经制定、评审并批准验收测试完成标准；
- 合同、附件规定的各类文档齐全。

验收测试的主要内容

- 新建系统产品是否运行正常，达到预定的目标；
- 各个子系统是否运行正常，达到预定的目标；
- 各个功能模块是否运行正常，达到预定的目标；
- 按照系统使用说明书上所说的方法去做能否实现；
- 按照系统维护手册上所说的方法去做，能否实现；
- 测试文档验收，测试过程文档是否齐全，可信，符合标准。
- 测试评估，从总体对测试的质量进行评估；
- 测试建议，对本次测试工作指出不足，需要在以后工作中改进的地方。

验收测试的设计思路

- 验收测试由3大部分组成
  - *软件配置审核*
  - *可执行程序测试*
  - *验收测试分为用户应用系统验收测试和 外包软件的验收测试*
- 验收测试的要点
  - *对文档进行审核*
  - *对源代码进行审核*
  - *对配置进行审核*
  - *对测试程序或脚本进行审核*
  - *对可执行程序进行测试*
  - *按照需求说明书*
  - *对系统进行评审*

验收测试安排

- 能否为了加快进度，将验收测试与系统测试合并或重叠?
  - *满足下列条件，合并验收测试与系统测试是有意义的:*
    - 用户代表实质性地参与了系统测试;
    - 系统测试的环境足够真实;
    - 验收测试用例是系统测试用例的一个子集
- 针对产品软件或Web应用面向成千上万的用户，如何安排验收测试?
  - *将用户分类，对每类用户选择合适的用户代表;*
  - *有些企业使用来自某些特定公司的用户担当用户测试员;*
  - *有些企业发布产品的Beta测试版，请各类用户试用或体验新产品，同时搜集用户的反馈信息*

验收测试流程

- 一个典型的验收测试流程
  - *开发方的项目经理代表项目组提出项目验收申请*
  - *客户方和监理方检查项目测试验收的前提条件是否具备:*
    - 检查系统测试是否符合要求
    - 检查是运行准备工作是否就绪
    - 检查所要求的项目档案是否齐备
  - *以客户方、监理方为主，三方共同编制验收测试计划;*
  - *三方评审和批准验收测试计划;*
  - *客户代表和监理方、开发方共同确定验收测试用例集*
  - *项目经理领导项目组，按照验收测试计划完成测试准备工作;*
  - *建立验收测试环境，安装系统，准备验收测试数据;*
  - *客户方、监理方进行系统试运行，运行测试用例集，记录测试结果;*
  - *如果发现缺陷，则在确认后项目组立即着手解决*
  - *客户方、监理方完成回归测试，确认缺陷已经修复、并已经关闭。*
  - *项目组在修复缺陷后，更新相关的项目文档;*
  - *客户方、监理方审查验收测试执行情况，起草验收测试报告;*
  - *客户方、监理方签署验收测试报告*

验收测试用例设计要点

- 验收测试用例应当在研发阶段测试用例的基础上重新组织和编写，而不能拿来直接使用
- 验收测试用例应客户需求相对应，具有面向客户的特点
- 设计过程中要把握客户的关注点并适当展示软件的独有特性
- 在验收测试中发现软件的缺陷或与需求存在偏差的地方应与客户保持良好的沟通共同确定修复和改进计划

应用系统验收测试

- 是系统开发机构向用户移交系统时履行的正式手续，也是用户对新系统的认可
- 验收报告
  - *引言*
  - *性能指标*
  - *系统评价*
- 鉴定工作程序和文档资料
  - *鉴定组织工作*
  - *鉴定测试报告主要内容*
  - *测试结论报告主要内容*
  - *鉴定书草案主要内容*
  - *研究报告和技术报告主要内容*
  - *向鉴定考核小组提供的审变材料和鉴定材料*
  - *鉴定会会议程序*

外包软件的验收测试

- 外包软件的验收测试事实上是根据应用对象或产品对象被分为：非正式验收测试和正式验收测试。外包软件的验收测试的策略通常建立在合同需求和公司标准的基础上