SOFTWARE TESTING

Chapter 2

Basic Concepts of Software Testing

School of Data & Computer Science
Sun Yat-sen University

Approaches & Technologies

OUTLINE

- 2.1 软件缺陷
- 2.2 软件测试概述
- 2.3 软件测试的过程和方法
- 2.4 基于软件生命周期的软件测试方法
  - 软件生命周期测试概述
  - 基于风险的软件测试
  - 软件生命周期的阶段测试
  - 软件生命周期测试工具平台
- 2.5 软件测试的分类与分级

软件生命周期测试概述

- 软件生命周期测试是按照软件开发生命周期划分软件测试阶段形成的软件测试方法，也称为基于 SDLC 的软件测试。
  - 将测试延伸到软件开发的需求分析、设计审查阶段。
    - 体现“尽早地和不断地进行软件测试”的原则。
  - 将“质量保证”的部分活动归为测试活动。
    - 确保对软件生命周期的每个阶段进行质量管理；
    - 通过测试手段实现各个阶段的质量保证。

软件生命周期测试概述

- IBM 的统计数据
  - 缺陷发现：大约 60 个缺陷/千行源代码
  - 缺陷根源：2/3 的缺陷产生在需求和设计阶段
  - 缺陷修复成本：
    - 在需求和设计阶段发现的缺陷修正的花费最小；
    - 修正系统测试阶段发现的缺陷，花费是以上的 10 倍；
    - 发布产品以后，修正缺陷的花费将达 100 倍。

软件生命周期测试概述

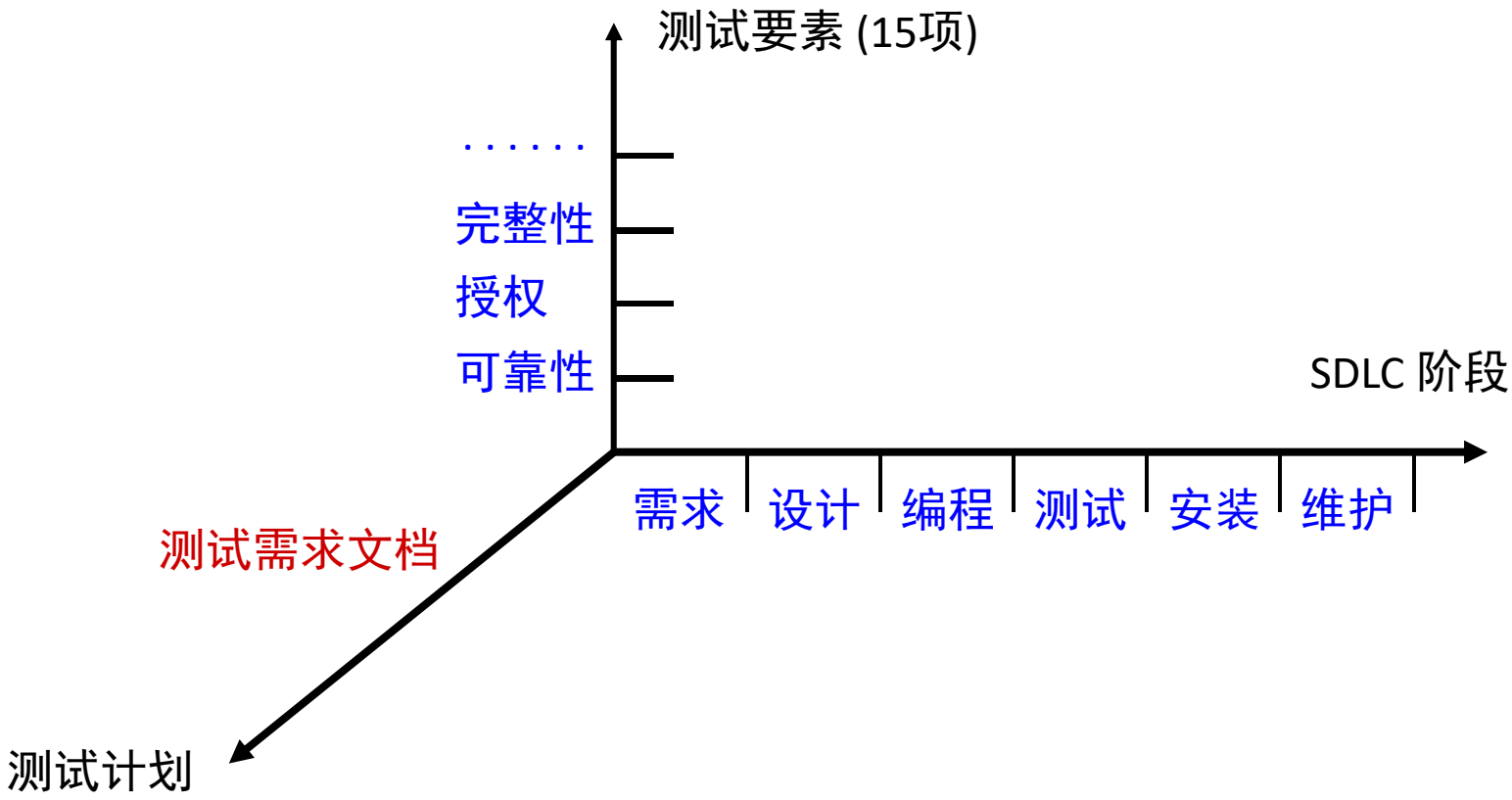
- 软件生命周期测试意味着软件测试与软件开发并行。
  - 软件生命周期测试伴随着整个软件开发周期，测试的对象不仅仅是程序，还包括了需求、功能和设计。
  - 测试与开发同步进行，在软件开发过程中持续地进行测试，有利于尽早地发现问题，尽早修正缺陷，降低测试成本，同时缩短项目的开发建设周期。
  - 软件生命周期测试需要系统化的管理 (工具) 支持。

- 软件生命周期测试概述
  - 软件开发生命周期各阶段对应的测试工作
    - 需求分析阶段
      - 审查需求分析文档、产品规格说明书，确认定义符合要求。
    - 设计阶段
      - 审查系统设计文档、程序设计流程图、数据流图等。
    - 代码编写阶段
      - 审查代码，检查代码是否遵守编程规范，验证程序实现了需求。
    - 测试和安装阶段
      - 检查实现的系统符合软件规格说明。
    - 维护阶段
      - 系统重新测试以确定改变的部分和未改变的部分能够继续工作。

软件生命周期测试概述

SDLC 阶段	验 证 活 动
需 求	确定验证的方法 确定需求的充分程度 生成功能测试数据 确定与需求符合的设计
设 计	确定设计的充分程度 生成结构和功能测试数据 确定设计与需求的一致性
编 程	确定编程实现的充分程度 生成各种程序/单元的结构和功能测试数据 确定编码与设计的一致性
测 试	确定测试计划的充分性 测试应用系统
安 装	把已经测试的系统部署运行
维 护	修改和重新测试

软件生命周期测试概述



软件生命周期测试概述

测试要素

测试要素的概念

- 测试要素描述测试的主要目标。
- 一个测试要素由若干个测试事件组成，用于验证该测试要素所描述的测试目标是否已经达成。
  - 一个测试事件描述了测试条件和可能发生的事件。
- 在 SDLC 的不同阶段，每一个测试要素所进行的测试内容有所不同，由不同的测试事件构成。

软件生命周期测试概述

测试要素

15项测试要素的解释 (测试目标)：

(1) 可靠性：系统在规定的时间内可以正常运行。

(2) 授权：特殊的授权可以执行一个特殊的操作。

(3) 文件完整性：文件被正确使用，恢复和存储的数据正确。

(4) 进程追踪：能够证实运行进程处在正常工作状态。

(5) 系统运行的连续性：发生非致命性问题后，系统仍然有能力继续运行关键的任务。

(6) 服务级别：系统有紧急情况发生时，程序的输出结果不经过处理或进行简单的处理后可以直接使用。

(7) 存取控制：防止系统被误用 (意外或者有意的)。

(8) 方法论：采用选择的方法论实现系统

软件生命周期测试概述

测试要素

15项测试要素的解释：

(9) 正确性：数据输入、过程处理和输出的正确性。

(9)′ 一致性：确保最终设计和用户需求完全一致。

(10) 易用性：多数人认为易于使用。

(11) 可维护性：出现问题时易于定位，并且进行修改。

(12) 可移植性：数据或者程序易于移植到其它系统上。

(13) 耦合性：系统中的组件易于联接。

(14) 性能：系统资源的占用率、响应时间、并发处理等能力。

(15) 易操作性：容易操作。

软件生命周期测试概述

测试要素

测试要素 (1-5) 在 SDLC 不同阶段的测试内容

	可靠性	授权	文件完整性	进程追踪	运行连续性
需求	建立精度等级	定义授权规则	定义文件完整性需求	定义重构处理需求	定义失效影响
设计	设计数据完整性控制	设计授权规则	设计文件完整性控制	设计审计追踪	设计中断计划
编程	实现数据完整性控制	实现授权规则	实现文件完整性控制	实现审计追踪	编写中断计划和过程
测试	人工、回归和功能测试	符合性测试	功能测试	功能测试	恢复性测试
安装	验证安装精度和完整性	禁止改变授权数据	检查产品文件完整性	记录安装审计追踪	保证以前测试的完整性
维护	修改精度要求	保存授权规则	保存文件完整性	修改审计追踪	修改中断计划

软件生命周期测试概述

测试要素

测试要素 (6-10) 在 SDLC 不同阶段的测试内容

	服务级别	存取控制	方法论	正确性	易用性
需求	定义希望的服务级别	定义系统的存取策略	按系统开发方法论定义需求	定义功能规格说明	定义易用性规格说明
设计	设计达到服务级别的方法	设计存取过程	按系统设计方法论设计系统	设计符合需求	设计便于实现易用性需求
编程	实现达到服务级别的系统	实现存取过程	按编程方法论编写程序	程序符合设计	程序符合设计使易用性最优化
测试	强度测试	符合性测试	按测试方法论执行测试	功能测试	人工支持测试
安装	实现故障安装计划	集成期间的存取控制	在产品环境中集成系统	正确的程序和数据的加入	传播易用性指令
维护	保存服务级别	保存安全级别	按系统维护方法论维护系统	修改需求	保存易用性

软件生命周期测试概述

测试要素

测试要素 (11-15) 在 SDLC 不同阶段的测试内容

	可维护性	可移植性	耦合性	性能	易操作性
需求	决定可维护的规格说明	决定可移植的要求	定义系统间的接口	建立性能准则	定义易操作要求
设计	设计是可维护的	设计是可移植的	设计考虑接口需求	设计能够实际达到这些准则	把要求转换为操作设计
编程	程序是可维护的	程序符合设计	程序符合接口设计说明	程序的实现能达到这些准则	程序符合设计
测试	检查	灾难性测试	功能和回归测试	符合性测试	操作测试
安装	文档齐全	文档齐全	调整接口	监控集成性能	实现操作过程
维护	保存可维护性	保存可移植性	保证正确的接口	保存性能级别	修改操作过程

- 软件生命周期测试概述
  - 测试计划
    - 测试计划的概念
      - 测试计划是一类描述测试活动的范围、方法、资源和进度的文档。
      - 测试计划确定测试项、被测特性、测试任务、任务执行者以及各种可能风险。
      - 测试计划可以有效预防可能风险，保障测试的顺利实施。
    - ISTQB 国际软件测试资质认证委员会
      - ISTQB (International Software Testing Qualification Board) 将测试过程分为计划和控制阶段、分析和设计阶段、实现和执行阶段、评估出口准则和报告阶段以及结束收尾阶段，分别解决“做什么”、“如何做”、“实施的具体步骤”、“发现缺陷并跟踪缺陷”、“评估测试报告”几个问题。

- 软件生命周期测试概述
  - 测试计划
    - 测试计划模板：IEEE 829
      - 模板内容包括组织形式 (强矩阵、平衡矩阵、弱矩阵)，测试对象，测试的需求跟踪和覆盖，测试的“通过/失败”标准，测试的挂起标准和恢复条件，工作的任务分配，项目可交付物。
      - http://zh.wikipedia.org/wiki/IEEE_829
      - https://en.wikipedia.org/wiki/Software_test_documentation
    - 制订测试计划常会遇到的问题
      - 测试计划未能在开发前期开始制订实行。
      - 测试计划的组织者缺乏对特殊应用软件的测试经验。
      - 测试的难度和复杂性可能太大，缺乏自动化工具，难以预先计划和控制。

- 软件生命周期测试概述
  - 软件生命周期测试的准入、准出条件
    - 开始软件生命周期测试的准入条件：
      - 具有测试合同 (或项目计划)；
      - 具有软件测试所需的各种文档；
      - 所提交的被测软件处于可控状态；
      - 软件源代码正确通过编译或汇编；
      - 能够从一开始介入被测软件的开发周期。

软件生命周期测试概述

软件生命周期测试的准入、准出条件

完成软件生命周期测试的准出条件

- 已按要求完成合同 (或项目计划) 所规定的软件测试任务；
- 实际测试过程遵循原定的软件测试计划和软件测试说明；
- 客观、详细地记录了软件测试过程和软件测试中发现的所有问题；
- 软件测试文档齐全、符合规范；
- 软件测试的全过程至始至终在控制下进行；
- 软件测试中的问题或异常有合理解释或正确有效的处理；
- 软件测试工作通过了测试评审；
- 全部测试软件、被测软件、测试支持软件和评审结果已纳入配置管理。

基于风险的软件测试

- 风险的概念
  - 风险可定义为事件、危险、威胁或情况等发生的可能性以及由此产生的不可预料的后果。
  - 产品风险与产品失效概率和失效严重程度 (预期损失) 相关。风险级别可以简化为：

    产品风险度 = 失效概率 (0-100%) × 预期损失 (0-10)。

    - 失效概率包括使用频率和缺陷概率两个参数。
      - 使用频率：含缺陷的产品组件的使用频率越高，缺陷被激活 (产品失效) 的概率就越大。
      - 缺陷概率：缺陷概率是指产品 (组件) 中包含缺陷的概率。产品的复杂度越高，缺陷概率就越大。

- 基于风险的软件测试
  - 风险的概念
    - 关于风险的一些观点
      - 风险是潜在的问题。
      - 风险是导致系统失败的原因，但不一定所有的风险都会导致系统失败。
      - 系统的风险始终存在，不能消除，但通过适当的手段可以降低风险发生的概率。
      - 风险的属性决定了风险测试的类型和测试方法。
    - 产品风险存在的原因
      - 测试团队需要在时间、成本和质量等各个方面进行平衡和协调，很难达到“零”缺陷的理想测试目标，由此带来了开发管理风险和产品质量风险。

- 基于风险的软件测试
  - 风险的概念
    - 系统风险列举：
      - 某部分产生错误导致的结果
      - 未被验证的数据交换被接受
      - 文件的完整性被破坏
      - 系统未能安全恢复 (完全恢复成备份时的状态)
      - 系统未能暂停运行
      - 进行维护工作时，系统性能下降到不能接受的水平
      - 系统的安全性未能保证
      - 系统的操作流程未符合用户的组织策略和长远规划
      - 系统的可靠性和稳定性未能保证
      - 系统的易用性和维护性未能保证
      - 系统的易连接性未能保证

基于风险的软件测试

- 基于风险的软件测试 (Risk-based Testing, RBT)
  - 风险测试概念的提出：*Boris Beizer*, Software Testing Techniques (软件测试技术), 1983-1990。
  - RBT 是一种以软件质量风险作为测试出发点和测试活动主要参考依据的测试方法。通过对项目质量相关信息的收集与分析处理，可有效地识别与划分、归类不同的风险域，再结合对测试粒度、方法及实施人员能力等方面的认识对测试做出优化安排。风险分析活动应用到对应的测试领域，产品风险的分析与评估的结果决定测试策略，以此指导整体测试过程。
    - RBT 最重要的观点是将软件的潜在风险作为决定测试策略和安排的依据和目标。

基于风险的软件测试

- 基于风险的软件测试 (Risk-based Testing, RBT)
  - *James Bach*, Heuristic Risk-based Testing (基于风险的启发式测试), 1995。
    - 列出一个风险列表；
    - 对每个风险进行分析和评估，确定风险级别；
    - 进行考察每项风险的测试；
    - 当风险消失而新的风险出现的时候，调整测试策略。
  - RBT 需要解决的主要问题
    - 确定测试的优先级；
    - 有效选择测试重点；
    - 有效配置测试资源；
    - 分析和评估测试的有效性等。

基于风险的软件测试

基于风险的软件测试 (Risk-based Testing, RBT)

RBT 的基本过程

(1) Analyze the requirements.

(2) Documents (SRS, FRS, Use cases) are reviewed. This activity is done to find and eliminate errors & ambiguities.

(3) Requirements sign-off is one of the risk-reduction technique for avoiding the introduction of late changes into the projects. Any changes to requirements after the documents are baselined would involve a change control process and subsequent approvals.

(4) Assess the risks by calculating the likelihood and impact each requirement could have on the project taking the defined criteria's like cost, schedule, resources, scope, technical performance safety, reliability, complexity, etc. into consideration.

(5) Identify the probability of failure and high-risk areas. This can be done using risk assessment matrix.

基于风险的软件测试

基于风险的软件测试 (Risk-based Testing, RBT)

RBT 的基本过程 (续)

(6) Use a risk register to list the set of identified risks. Update, monitor and track the risks periodically at regular intervals.

(7) Risk profiling needs to be done at this stage to understand the risk capacity and risk tolerance levels.

(8) Prioritize the requirements based on the rating.

(9) Risk-based test process is defined.

(10) Highly critical and medium risks can be considered for mitigation planning, implementation, progress monitoring. Low risks can be considered on a watch list.

(11) Risk data quality assessment is done to analyze the quality of the data.

(12) Plan and define test according to the rating.

基于风险的软件测试

基于风险的软件测试 (Risk-based Testing, RBT)

RBT 的基本过程 (续)

(13) Apply appropriate testing approach and test design techniques to design the test cases in a way that the highest risks items are tested first. High-risk items can be tested by the resource with good domain knowledge experience.

(14) Different test design techniques can be used for e.g. using the decision table technique on high-risk test items and using 'only' equivalence partitioning for low-risk test items.

(15) Test cases are also designed to cover multiple functionalities and end to end business scenarios.

(16) Prepare test data and test conditions and test bed.

(17) Review the Test plans, Test Strategy, Test cases, Test reports or any other document created by the testing team.

(18) Peer review is an important step in defect identification and risk reduction.

基于风险的软件测试

基于风险的软件测试 (Risk-based Testing, RBT)

RBT 的基本过程 (续)

(19) Perform dry runs and quality checks on the results.

(20) Test cases are executed according to the priority of the risk item.

(21) Maintain traceability between risk items, tests that cover them, results of those tests, and defects found during testing. All testing strategies executed properly will reduce quality risks.

(22) Risk-based testing can be used at every level of testing, e.g. component, integration, system, and acceptance testing.

(23) At the system level, we need to focus on what is most important in the application. This can be determined by looking at the visibility of functions, at frequency of use and at the possible cost of failure.

(24) Evaluation of exit criteria. All high-risk areas fully tested, with only minor residual risks left outstanding.

(25) Risk-based Test Results reporting and metrics analysis.

基于风险的软件测试

基于风险的软件测试 (Risk-based Testing, RBT)

RBT 的基本过程 (续)

(26) Reassess existing risk events and new risk events based on Key Risk Indicators.

(27) Risk register updating.

(28) Contingency plans- This works as a fallback plan/emergency plans for the high exposure risks.

(29) Defect analysis and defect prevention to eliminate the defects.

(30) Retesting and Regression testing to validate the defect fixes based on pre-calculated risk analysis and high-risk areas should be most intensively covered.

(31) Risk-based automation testing(if feasible)

(32) Residual Risk calculation.

(33) Risk Monitoring and Control.

- 基于风险的软件测试
  - 基于风险的软件测试 (Risk-based Testing, RBT)
    - RBT 的基本过程

(34) Exit Criteria or completion criteria can be used for different risk levels. All key risks have been addressed with appropriate actions or contingency plans. Risk exposure is at or below the level agreed to as acceptable for the project.

(35) Risk profiling reassessment and customer feedback.

软件生命周期的阶段测试

- 全生命周期中软件测试的最终要求是：
  - 保证软件系统在全生命周期中每个阶段的正确性，验证在整个软件开发周期中各个阶段的软件质量是否合格。
  - 保证最终系统符合用户的要求和需求，确认最终交付给用户的系统是否满足用户的需求。
  - 用样本测试数据检查系统的行为特性。
  - 把尽可能多的问题在产品交给用户之前发现并改正。

- 软件生命周期的阶段测试
  - 需求阶段
    - 经典的软件工程理论中，软件测试是在代码完成后开始的。全生命周期软件测试要求从软件的需求定义开始进行需求测试。
    - 软件工程统计结果发现：
      - 50% 以上的系统错误是由错误的需求或缺少需求所导致。
      - 超过 80% 的开销用在追踪需求错误上。
        - 追踪需求错误的过程是低效的。
    - 需求测试贯穿整个软件开发周期，对软件生命周期测试的各个阶段起着指导作用。

- 软件生命周期的阶段测试
  - 需求阶段
    - 只有具备软件需求测试，软件的测试、验证、确认才有意义。
      - 确保大部分软件缺陷不会进入到设计和编码阶段
        - 在这个阶段所有的花费都是值得的
    - 准备风险列表
      - 确定风险
        - 风险分析
        - 风险检查表
      - 建立控制目标
      - 确定有足够的控制力度

- 软件生命周期的阶段测试
  - 需求阶段
    - 分析测试要素
      - 需求的设计是否遵循已定义的方法
      - 已经提交了定义好的功能说明
      - 预先定义了系统界面
      - 预先估计了性能标准
      - 预先估计了容忍度
      - 预先定义了权限规则 (存取控制)
      - 预先定义了文件完整性
      - 预先定义了需求的变更流程
      - 预先定义了失败的影响

- 软件生命周期的阶段测试
  - 需求阶段
    - 测试目标
      - 需求正确表达了用户的需要
      - 需求已经被定义和文档化
      - 需求的控制被明确化
      - 有合理的流程可遵循
      - 有合理的方法可供选择
      - 花费和收益成正比
    - 测试活动
      - 彻底分析需求的充分性，生成基础测试用例。
      - 确定哪些需求是可测试的，舍去含糊的、不可测试的需求，建立产品的需求并确认。

- 软件生命周期的阶段测试
  - 设计阶段
    - 设计人员根据需求分析详细定义要交付的产品，包括
      - 输入说明、过程说明、文件说明、输出说明、控制说明、系统流程图、硬件和软件的需求、操作手册说明书、数据保留的策略等等。
    - 测试任务
      - 分析测试要素
      - 给测试要素打分
      - 对设计进行评审
      - 检查修改的部分 (出现回归时)

- 软件生命周期的阶段测试
  - 设计阶段
    - 测试活动
      - 概要设计阶段
        - 阐述测试方法和测试评估准则，编写测试计划，成立测试小组，安排具有里程碑的测试日程。
      - 详细设计阶段
        - 开发/获取支持工具，生成功能测试数据和测试用例。
        - 检查设计中遗漏的情况、错误的逻辑、不匹配的模块接口、不合理的数据结构、错误的 I/O 假定、不够充分的用户界面等等。

- 软件生命周期的阶段测试
  - 设计阶段
    - 设计阶段的评审
      - 设计阶段的评审是对阶段处理成果的完整性进行正式的评价；风险越高的项目需要越详细的设计评审。
      - 评审前
        - 为评审分配足够的时间，成立评审组，并对组员进行培训。
      - 评审中
        - 评审组和项目组一起进行评审，并且只对文档进行评审。
      - 评审后
        - 将评审的结果写成正式报告。

- 软件生命周期的阶段测试
  - 设计阶段
    - 设计阶段工具的应用
      - 设计阶段使用静态和动态测试工具测试系统的结构。评分工具和设计评审工具是广泛使用的两种测试工具。
        - 评分工具用于标识风险，根据得分的结果确定系统的风险程度。
        - 设计评审工具用于对实际阶段处理的完整性进行正式的评价，它是测试设计规格说明的工具。
      - 测试小组对设计进行检查，所涉及的项目包括：
        - 遗漏的情况；错误的逻辑；不匹配的模块接口；不合理的数据结构；错误的 I/O 假定；不够充分的用户界面等等。

- 软件生命周期的阶段测试
  - 设计阶段
    - 测试要素打分
      - 设计了对数据完整性的控制
      - 设计了权限规则 (存取控制)
      - 设计了对文件完整性的控制
      - 设计了审计追踪
      - 设计了发生意外情况时的计划
      - 设计了如何达到服务水平的方法
      - 定义了权限流程
      - 定义了完整的方法学
      - 设计了保证需求一致性的方法
      - 进行了易用性的设计
      - 设计是可维护的、简单的

- 软件生命周期的阶段测试
  - 设计阶段
    - 测试要素打分 (续)
      - 交互界面设计完毕
      - 定义了项目成功的标准

软件生命周期的阶段测试

- 编码阶段
  - 在编码阶段完成测试用例开发，对程序进行实际测试。
    - 编码阶段已经有许多测试工具得到开发应用，比如编码走查和检查、静态分析和动态测试技术等。
  - 测试任务
    - 编码阶段测试需要解决的首要问题，是编码是否和设计一致。其次是：
      - 系统是否可维护
      - 是否正确实现了系统的规格说明
      - 编码是否按照既有的标准 (规范) 进行
      - 是否有充分的测试计划用于评价可执行的程序
      - 源程序是否提供了足够的文档资料
      - 源程序内部是否有足够的注释等

- 软件生命周期的阶段测试
  - 编码阶段
    - 编码阶段测试完成后形成下列输出
      - 编码说明书
      - 程序文档
      - 程序列表
      - 可执行的程序
      - 程序流程图
      - 操作介绍 (手册)
      - 单元测试结果

- 软件生命周期的阶段测试
  - 编码阶段
    - 测试关注点
      - 完成对数据和文件完整性的控制
      - 定义授权的规则
      - 实现审计追踪
      - 对意外情况发生后的处理计划做了规划
      - 对系统如何达到预定义的服务水平做了计划
      - 完成了对安全问题的处理流程
      - 编码工作是依据规定的方法完成的
      - 编码与设计相一致 (正确性，易用性，简洁性，耦合性)
      - 代码是可维护的
      - 已开发了操作流程
      - 定义了程序成功的性能标准

- 软件生命周期的阶段测试
  - 测试阶段
    - 测试任务
      - 测试阶段要进行第三方的正式确认测试，检验系统是否按照用户的要求运行；用户能够成功地安装一个新的应用系统来进行测试。
    - 测试关注点
      - 前期文档
      - 测试阶段的测试计划
      - 测试用例
      - 前期测试的测试结果
      - 第三方测试反馈
      - 正式的测试总结报告

- 软件生命周期的阶段测试
  - 测试阶段
    - 典型测试类型
      - 手册与文档测试 (易用性)
      - 一致性测试 (授权, 安全性, 性能)
      - 功能点测试 (完整性、 正确性，审计，追踪)
      - 覆盖性测试
      - 压力测试
      - 依照预先定义的测试方法
      - 可维护性测试
      - 灾难性测试
      - 功能和回归测试
      - 易操作性测试

- 软件生命周期的阶段测试
  - 测试阶段
    - 测试方法
      - 测试用例
        - 如何建立有效的测试用例
      - 测试文件
        - 测试用例应当包含合法的和非法的输入
        - 每一个动作只进行一次关键操作
      - 测试数据
      - 结果分析
      - 测试工具
        - 自动化测试工具
        - 压力测试工具等

软件生命周期的阶段测试

- 测试阶段
  - 测试活动
    - 进行第三方的正式确认测试，检验所开发的系统是否能按照用户提出的要求运行。
    - 用户成功安装应用系统后进行的测试：
      - 功能测试：运行部分或全部系统，确认用户的需求被满足。
      - 符合性测试：验证软件系统与相应标准的符合程度。
      - 强度测试：测试系统对极端条件的反应，标识软件的薄弱点，指出系统能够经受的正常的工作量。
      - 性能测试：通过测量响应时间、CPU 使用率和其它量化的操作特征，评估软件系统的性能指标。
      - 操作测试：在没有开发人员的指导和帮助下，由操作人员进行测试，以评估操作命令的完整性和系统的易操作性。
      - 恢复测试：故意使系统失败，测试人工和自动的恢复过程。

- 软件生命周期的阶段测试
  - 测试阶段
    - 测试报告
      - 目标
        - 呈现目前项目的实际状况
        - 明确指出测试完成的内容
        - 给出系统操作性能的评价
        - 明确系统可以进行产品化工作的时机
      - 关注点
        - 测试报告是否有效、有用
        - 测试的信息是否充分
        - 测试状况是否能真实反应系统的状况

- 软件生命周期的阶段测试
  - 测试阶段
    - 测试报告 (续)
      - 测试期间数据的收集
        - 有关测试结果的积累数据
        - 有关测试任务、测试集合和测试事件的描述
        - 缺陷分析 (严重的缺陷、缺陷类型、是否存在没有发现的缺陷、缺陷的原因)
        - 测试效果评估

- 软件生命周期的阶段测试
  - 测试阶段
    - 测试总结报告
      - 报告目前的软件状态
        - 功能/测试矩阵
        - 功能测试的状态报告、侧重点分析
        - 关于功能的工作时间轴
        - 期望发现缺陷数与实际发现缺陷数的比率
        - 没有发现的缺陷数和改正的缺陷数的差距
        - 按照类型分类，没有改正的缺陷数的平均值
        - 缺陷分类报告
        - 测试活动报告
    - 各个阶段的项目测试总结报告：系统测试报告、确认测试报告

- 软件生命周期的阶段测试
  - 安装阶段
    - 测试准备
      - 安装计划
      - 安装流程图
      - 安装文件和程序清单
      - 对安装程序进行测试，给出测试结果
      - 将程序运行的软硬件要求放入产品说明中
      - 对于新操作人员的使用说明书
      - 对于新使用者的操作说明和操作流程
      - 安装过程中的各项可能发生的结果的说明

- 软件生命周期的阶段测试
  - 安装阶段
    - 测试关注点
      - 对程序安装的正确性和完整性进行核对
      - 校验产品文件的完整性
      - 安装的审查、追踪被记录
      - 安装之前该系统已经被证实没有问题
      - 如果安装失败，系统有相应的解决方案
      - 安装过程进行了权限控制 (安全性)
      - 安装遵循一定的方法、步骤
      - 需要的配套程序和数据已经放进了产品中
      - 已交付使用说明
      - 相关文件已经完整 (可维护性)
      - 接口已经被合理调整 (耦合性)
      - 综合的性能达到了用户要求

- 软件生命周期的阶段测试
  - 安装阶段
    - 测试检查表的使用
      - 使用测试工具建立和管理检查表
      - 选择测试的范围
      - 选择检查表
      - 理解检查表中的问题的意义
      - 提前测试用户的检查表
      - 使用该检查表模拟运行一遍
      - 记录有用的信息
      - 评估检查表和检查流程

- 软件生命周期的阶段测试
  - 安装阶段
    - 测试标准
      - 检查数据的正确性
      - 校验检查表和产品的正确性
      - 使用测试标准去检验发生的问题
    - 测试活动
      - 安装阶段测试是测试应用系统的安装过程，分两种类型：
        - 验证安装程序的正确功能。
        - 验证安装过程的性能。
      - 安装通常应该在一个比较短的时间跨度范围内完成，而不是需要花1小时或几个小时。

- 软件生命周期的阶段测试
  - 验收阶段
    - 定义用户角色
      - 确定最终用户的范围
      - 确认临时的和最终产品的验收标准
      - 计划每一个验收过程由谁负责和如何执行
      - 计划资源分配
      - 计划时间分配
      - 准备验收计划
      - 为每一项验收工作给出结论

- 软件生命周期的阶段测试
  - 验收阶段
    - 确定验收标准
      - 功能上
      - 性能上
      - 接口质量上
      - 过载后的软件质量
      - 安全性
      - 软件的稳定性

- 软件生命周期的阶段测试
  - 验收阶段
    - 编写验收计划
      - 项目描述
      - 用户职责
      - 行政流程
      - 验收活动描述
      - 每一个验收项的评审
      - 最终的验收测试步骤

- 软件生命周期的阶段测试
  - 验收阶段
    - 执行验收计划
      - 验收测试和评审
    - 验收的结果
      - 典型的验收结果
        - 在进入下一个活动之前问题或者变更必须被接受
        - 工作可以继续，但是下次评审之前必须更正。
        - 没有任何的更改

- 软件生命周期的阶段测试
  - 维护阶段
    - 工作重点
      - 测试和培训
    - 目标：
      - 开发一些测试用例，预先发现一些问题。
      - 在运行情况发生变化后，预先修正一些错误。
      - 编写必要的培训材料
      - 对有关的人员进行培训
      - 同用户进行接触

- 软件生命周期的阶段测试
  - 维护阶段
    - 开发更新测试计划
      - 测试计划简短，必须能够在短时间内完成。
      - 只测试变化的部分
      - 测试要素
        - 变化的数据交换
        - 变化的程序
        - 操作流程
        - 用户的操作习惯
        - 不同系统之间的互联
        - 语言版本
        - 安全性
        - 备份/恢复

- 软件生命周期的阶段测试
  - 维护阶段
    - 编制培训计划
      - 对系统进行概览
      - 对系统假定一些错误，给出处理方法
      - 培训材料
        - 对项目内容的陈述
        - 用户使用方法
        - 对错误列表上的问题给出解释
        - 对报告进行解释，并且说明如何使用 (图标、数据等)。
        - 对输入数据进行解释

- 软件生命周期的阶段测试
  - 维护阶段
    - 反馈
      - 反馈有利于提高软件水平
      - 反馈包括：用户反馈和测试反馈
      - 反馈分类：错误反馈和建议反馈
      - 反馈的内容：
        - 测试的数量和内容
        - 发现的问题数量和分类
        - 技术上的问题
        - 应用上的问题
      - 反馈信息整理后加入到相关的测试数据中

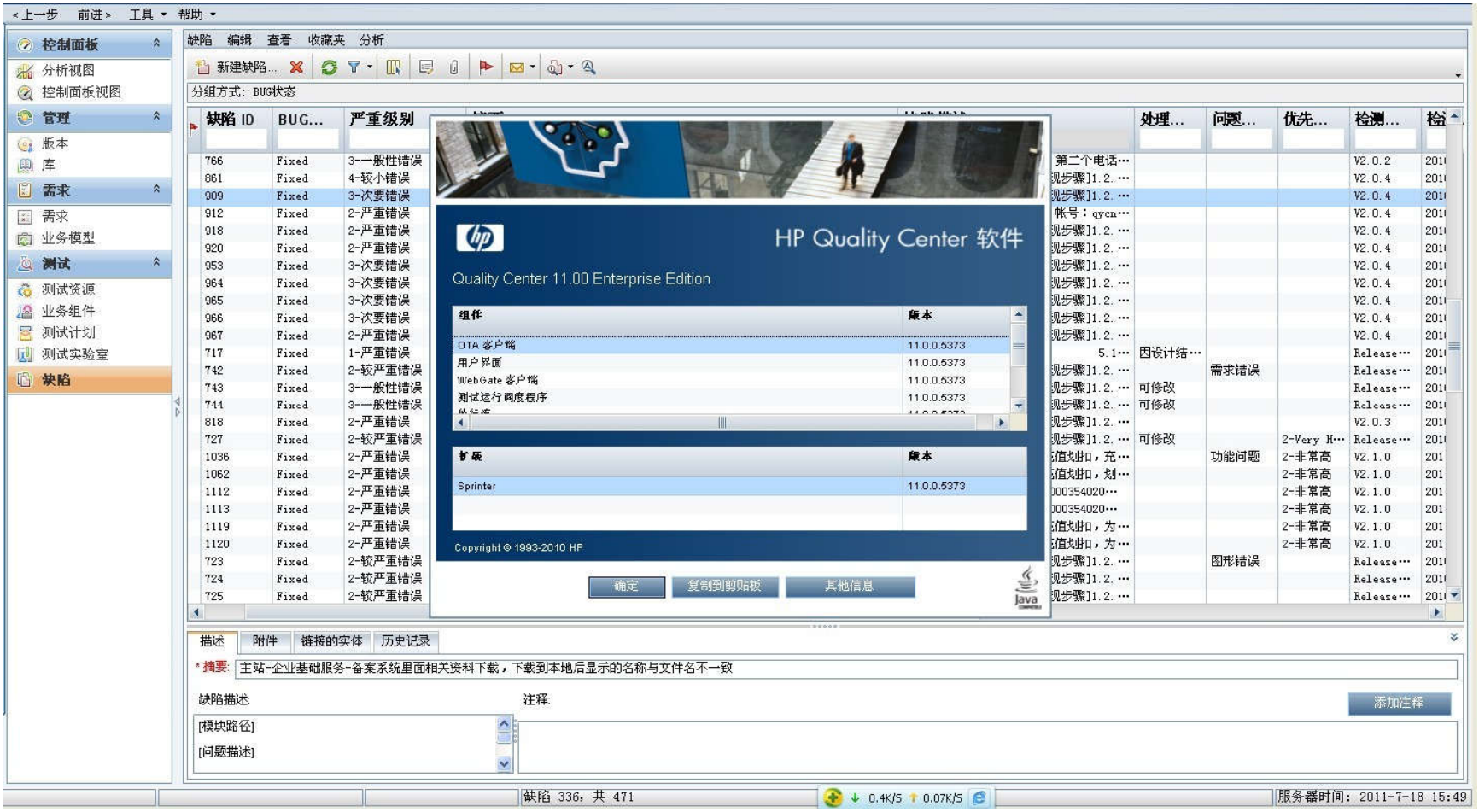
- 软件生命周期的阶段测试
  - 维护阶段
    - 测试活动
      - 在维护阶段，每当软件发生变化时应同时进行测试，目的是保证系统的变化在操作环境中能正确的运行。
      - 在维护阶段进行回归测试，重新运行以前进行过的测试，消除由于软件修改而带来的各种错误。
      - 主要的测试元素有：
        - 变化的事务、变化的程序、运行过程、控制组过程、系统内的连接、作业控制语言、软件系统界面、安全、备份/恢复过程等等。

- 软件生命周期测试工具平台
  - 惠普应用生命周期管理 (HP ALM)
    - 概述
      - HP ALM 使从业人员能够管理应用生命周期，并对生命周期进行测试。这些测试是从项目建议到运营全过程中贯穿应用交付。
      - HP ALM (release 12.53) 是一个以任务为导向的系统，可在应用交付过程中支持各参与方，并与主要开发工具整合，实现团队内和不同团队间的工作流程自动化，强化并加速了应用生命周期管理和各阶段的测试。

基于软件生命周期的软件测试方法

软件生命周期测试工具平台

惠普应用生命周期管理 (HP ALM)



- 软件生命周期测试工具平台
  - 惠普应用生命周期管理 (HP ALM)
    - HP ALM 为客户带来的优势：
      - HP ALM11“项目规划及跟踪”建立了发布标准，并在整个流程中基于实时监测来管理发布进程及准备情况。
      - 需求、开发及质量工具间的三路追踪，实现了对应用变化的快速组内分析及执行。
      - 通过惠普敏捷加速器 (HP Agile Accelerator 4.0)，灵活地按项目类型 (瀑布式项目、定制项目、敏捷项目) 支持优化交付方式，赠送的基础版本以及付费的高级版本均有此功能。
      - 减少因应用故障导致的业务风险，这些故障源于由混合以及富互联网应用引起的在功能、性能和安全方面的缺陷。
      - 通过易于发现、重复利用以及分享关键应用工具 (包括需求、测试及缺陷检测)，降低成本并缩短交付时间。

- 软件生命周期测试工具平台
  - 惠普应用生命周期管理 (HP ALM)
    - HP ALM 为 HP Quality Center/HP Performance Center 打下基础。
      - 这些解决方案能简化和自动化应用质量和性能验证，从而降低运营成本，并将更多的资源投入到新应用及服务中。
      - 借助 HP Sprinter，通过自动化手动测试 (如数据创建，以及在多环境下进行的重复手动测试) 来加速应用部署。
      - 通过 HP TruClient 改善测试创建。TruClient 是 HP Load Runner 11.0 的一部分，主要对应用性能进行测试，无需执行耗时的脚本处理。
      - 借助 HP Unified Functional Testing 11.0 为复合应用提供单一自动化解决方案，降低 GUI 和非 GUI 测试应用功能故障。
      - 该方案由惠普功能测试 (HP Functional Test) 和惠普服务测试 11 (HP Service Test 11) 组成。

- 软件生命周期测试工具平台
  - 惠普应用生命周期管理 (HP ALM)
    - HP ALM 能够帮助用户组织和管理应用程序生命周期管理过程的所有阶段。
      - 指定版本：制定一个发布周期管理计划，更高效地管理应用程序发布和周期。
        - 追踪应用程序发布，并根据计划确认发布是否正常。
      - 指定需求：分析应用程序并确定需求。
        - 可以跨多个发布和周期管理需求，并在需求、测试、缺陷之间实现多维追踪。
        - ALM 为需求覆盖和关联到质量评估和商业风险中的缺陷提供实时可见的功能。
      - 计划测试：创建一个基于需求的测试计划。
        - ALM 为手动和自动测试提供了知识库。

- 软件生命周期测试工具平台
  - 惠普应用生命周期管理 (HP ALM)
    - HP ALM 能够帮助用户组织和管理应用程序生命周期管理过程的所有阶段。(续)
      - 执行测试：创建测试集，完成测试运行。
        - ALM 支持健壮测试、功能测试、回归测试以及更高级测试。根据计划执行测试，从而识别和解决问题。
      - 追踪缺陷：报告在应用程序中检测到的缺陷，跟踪修复进程。
        - ALM 通过分析缺陷和缺陷趋势，帮助做出合理的“执行/不执行”决策
        - ALM 支持完整的缺陷生命周期 - 从初始问题检测到修复缺陷以及确认缺陷修复。
      - 在每个阶段通过生成的详细报告和图表对数据进行分析。

- 软件生命周期测试工具平台
  - 一些主流生命周期测试管理平台
    - HP ALM (QC11)
      - Java EE 架构，基于 Windows、Linux、Solaris 等系统，Web 服务器 Apache、IIS，应用服务器 JBOSS、WebLogic、WebSphere；支持企业级用户，支持高并发
      - Mercury TestDirector
        - 早期产品
        - B/S 架构，基于 Windows 2000/IIS4.0，数据库最低可以是Access，内存 128M，CPU Pentium Ⅱ以上

- 软件生命周期测试工具平台
  - 一些主流生命周期测试管理平台
    - Atlassian Jira
      - 敏捷&项目管理模式
      - 测试需求和测试用例和其它工具集成
      - JavaScript 类库实现，纯 B/S 架构
      - 支持移动端访问
    - Rally
      - 敏捷&项目管理模式
    - Micro Focus SCTM (Silk Central Test Manager)
      - 良好的开放性
      - JavaScript 类库实现，纯 B/S 架构
      - 支持移动端访问

- 软件生命周期测试工具平台
  - 一些主流生命周期测试管理平台
    - 51Testing TestPlatform
    - IBM RQM

Thank you!