试卷结构

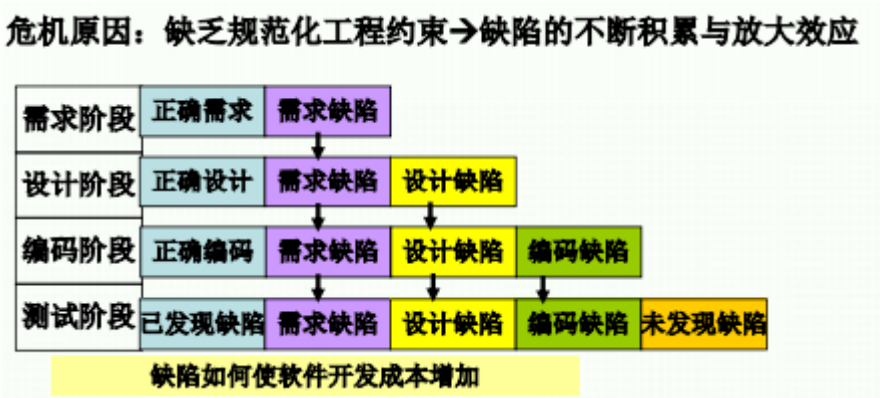
1. 选择题1×10
2. 填空题1×20
3. 简答题5×4
4. 设计题4-5题 50分

软件测试概述

软件测试的产生背景

- 软件危机：质量、维护费用、进度、成本、人员、局面失控

缺陷累积放大



缺陷出现的原因

1. 需求说明书（主要原因）：随意、易变、沟通不足
2. 设计（次要原因）：随意、易变、沟通不足
3. 编码：软件复杂度、进度压力、低级错误
4. 其他：理解错误、测试错误

有关测试观点的正确理解

1. 测试是为了证明程序有错，而不是证明程序无错误
2. 一个好的测试用例是在于它能发现至今未发现的错误
3. 一个成功的测试是发现了至今未发现的错误的测试

软件测试的基本概念

测试定义

分析某个软件项以发现现存和要求的条件之差别并评价此软件的特性

测试与调试

1. 软件测试是找出软件已经存在的错误,而调试是定位错误,修改程序以修正错误.
2. 软件测试从一个已知的条件开始,有预知的结局 而调试从未知的条件开始,其结局不可预知
3. 软件测试可以计划,可以预先制定测试用例和过程,工作进度可以度量.而调试不能计划,进度不可度量.
4. 调试是在测试之后,在方法,思路,策略上都有所不同.
5. 测试的对像可以是文档和代码 而调试的对像只能是代码
6. 调试是随机性的 由程序员完成 为了程序可运行;测试是有目的性的 由测试人员完成 为了程序可完成指定功能

测试目的

1. 确保软件质量
2. 确保软件开发过程方向的正确性

测试原理/原则

1. 用户需求至上
2. 测试是有计划的活动
3. 缺陷出现的集群性
4. 测试应从“小规模”走向“大规模”
5. 穷尽测试不可能
6. 有效的测试应由第三方独立进行
7. 测试无法揭示所有缺陷
8. 测试的杀虫剂悖论：潜在测试对已进行的测试具有免疫力
9. 测试是有风险的行为
10. 并非所有的软件缺陷都需要修复

测试用例（三要素）

输入、执行条件、期望输出

软件测试类型

视角1：测试技术
1. 白盒测试　　2. 黑盒测试　　3. 灰盒测试

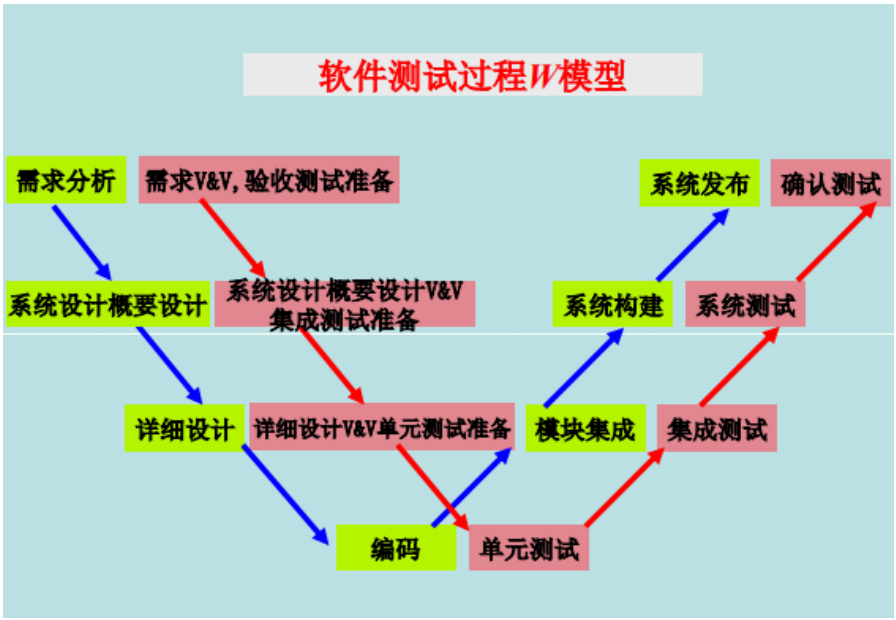
视角2：开发阶段
1. 单元测试　　2. 集成测试　　3. 确认测试　　4. 系统测试
5. 性能测试　　6. 回归测试　　7. 验收测试

视角3：执行状态
1. 静态测试　　2. 动态测试

视角4：执行主体
1. 开发方测试　　2. 用户测试　　3. 第三方测试

视角5：特殊测试
1. 国际化测试　　2. 即兴测试　　3. 兼容性测试　　4. 安全性测试
5. 可用性与易获得性测试　　6. 面向对象系统测试　　7. Web测试

软件测试过程W模型



软件测试现状和趋势

45%

白盒测试

白盒测试基本概念（定义、意义、实施者、步骤）

基于源程序或代码结构

静态、动态

主要的单元测试方法

实施者

单元测试阶段：开发人员 集成测试阶段：测试人员和开发人员

步骤

动态：程序逻辑分析；生成测试用例；执行测试；分析覆盖标准；判定测试结果

静态：桌面检查；代码走查；代码审查

静态白盒测试

不执行代码，审查软件设计、体系架构和代码（架构化分析）

桌面检查

代码编写者，编译执行前

1. 无结构化或形式化方法保证
2. 不维护记录或检查单

行号 变量 条件 输入输出

优点：

1. 编译者容易理解和阅读自己代码
2. 开销小，没有指定进度
3. 尽早发现缺陷

缺陷：

1. 开发人员不是实施测试的最佳人选
2. 依靠个人勤奋和技能，有效性难以保证

代码评审 Inspection

最正式的审查类型，高度组织化

计划、概述、准备、审查会议、审查报告、返工、跟进

优点：

1. 发现代码缺陷
2. 提高代码质量
3. 及早定位缺陷群集位置

缺点：

1. 费时间
2. 很多人
3. 不能保证参与者全部理解程序

动态白盒测试

基本路径测试（流图）

优势：

1. 更细的控制流描述——程序流图

   同时刻画判定和条件对程序控制流的影响；判定条件覆盖
2. 更少的覆盖路径——基本路径

圈复杂度的计算

1. 域的数量
2. E-N+2
3. P+1（p为判定节点数）

基于控制流覆盖的测试

覆盖能力

语句——判断——条件——判断条件——条件组合——路径

覆盖条件

1. 语句覆盖测试——语句覆盖：每条语句都执行一遍

   100%的语句覆盖很困难
2. 条件测试——判定覆盖（每个判断TF）、条件覆盖（每个判断中的每个条件）、判定条件覆盖（每个条件和由条件组成的判断）、条件组合覆盖（条件的取值组合）
3. 路径测试——路径覆盖（所有可能的路径）

- 短路、尽量少的测试用例、不用流图

MC/DC覆盖

满足判定条件覆盖的基础上满足判断中的每个条件对判断的取值是独立的

数据流测试

程序图/流图

循环测试

1. 简单循环：跳过、一次、两次、m次(m<n)、n-1,n,n+1次
2. 嵌套循环：最内层（外层设为最小）、上一层（外层取最小值，内层取典型值）、同时取最小或同时取最大
3. 串接循环：独立（简单循环）or不读立（嵌套循环）
4. 非结构循环：结构化

白盒测试工具

黑盒测试

黑盒测试基本概念

定义、意义、**目的、实施者**、步骤、**进入退出条件**

定义：基于规格说明书，不要求考察代码，用户视角。

意义：有助于软件产品的总体功能验证：

1. 检查明确需求和隐含需求
2. 采用有效输入和无效输入
3. 包含用户视角

实施者：测试人员

目的：测试软件的功能和非功能性需求

步骤：规格说明书；测试用例；执行测试；判断结果

进入退出条件：有可测的功能；计划完成或成本超出

黑盒测试方法基础（基于需求、正面测试和负面测试）

代码不可见的前提下面对未知的缺陷，如何设计测试用例：

1. 随机测试 缺点：缺陷分布不均匀；随机测试很难取到特定值
2. 基于需求规格说明书设计测试用例，既要有证明测试，也要有负面测试 基于需求的测试 目的：确认软件需求规格说明书列出的需求 RTM需求跟踪矩阵
3. 可跟踪每个需求的测试状态而不会遗漏任何需求
4. 优先执行优先级高的测试用例，尽早发现高优先级区域内缺陷
5. 可导出特定需求对应的测试用例清单
6. 评估测试工作量和测试进度的重要数据 正面测试用于验证已知测试条件，证明软件可以完成所期望的工作

- 注意：预期产品给出一个错误时它确实给出该错误，这也是正面测试的一部分 负面测试用于通过未知条件把产品搞垮

等价划分

等价划分方法步骤： Step1:选择划分准则（范围、取值、布尔、集合...） Step2:根据准则确定有效等价类和无效等价类 Step3:从等价类中选取样本数据 Step4:根据需求写预期结果 Step5:加入特殊值 Step6:执行测试

边界值分析

软件的两个主要缺陷源：（1）条件；（2）边界 条件：变量取值需要采取的特定行动 边界：各种变量值的“极限” 边界值分析： 能有效捕获出现在边界处的缺陷的一种测试方法；利用并扩展了缺陷更容易出现在边界处的概念 缺陷出现在边界处的原因：（1）使用比较操作符时未仔细分析（2）多种循环和条件检查方法引起的困惑（3）对边界附近需求的理解不够

Paul Jorgensen公式：

1. 4n+1：基本边界测试
2. 6n+1：健壮性边界测试
3. 3m：边界条件测试 n: 存在边界值的参数个数 m: 边界值条件数

因果分析

关系：恒等、非、或、与、

输入约束：互斥E、包含I、唯一O、要求R

输出约束：屏蔽M：当E1是1时，E2必须是0；当E1是0，E2的值不定。

因果分析法总结

Step1:分析规格说明书，识别原因和结果 Step2:在因果图连接原因和结果 Step3:标明原因之间以及结果之间的约束条件 Step4:因果图转换为因果图列表进而生成决策表 Step5:决策表的规则转换为测试用例

决策表

决策表并非因果图的一个辅助工具

决策表构造 Step1:列出所有的条件桩和动作桩 Step3: 填入条件项 Step4: 填入动作项，得到初始决策表 Step5：简化决策表，合并相似规则

(1)决策表中的条件——测试用例的输入条件 (2)决策表中的行动——测试用例的期望输出 (3)一条决策规则——一个测试用例

正交数组

记法：$L_{行数}(水平数^{因子数})$

一、正交表的构成 因子(Factor):欲考察的变量因素（输入参数） 水平(Levels）: 单个因子的取值（输入取值） 因子数(Factors):正交表中列的个数 水平数(Levels):单个因子的取值个数 行数(Runs):正交表行数，即测试用例个数

整齐可比性： 在同一张正交表中，每个因子的每个水平出现的次数是完全相同的。

均衡分散性： 在同一张正交表中，任意两列（两个因子）的水平搭配（横向形成的数字对）是完全相同的。

正交数组测试步骤 Step1: 确定因子和水平 有哪些因子？总因子数是多少？每个因子有哪些取值？其水平数是多少？ Step2: 判断是否能使用正交数组 因子的个数少，如只有2个，则不适用；因子的个数较多时可考虑正交数组 Step3: 选择合适的正交表 根据因子数和水平数，选择测试次数最少的正交表

Step4: 把变量值映射到表中 填入输入变量的取值 Step5: 正交测试用例制作 每一行对应一个测试用例 Step6: 补充测试用例 根据需求规则说明书，增加一些正交表没有覆盖但需要测试的用例

蜕变测试

随机测试

黑盒测试工具：测试工具原理、作用

自动化测试工具原理（以基于GUI的自动化测试工具为例）：运行被测软件的同时，捕获过程中的键盘、鼠标操作，生成脚本文件，这个脚本文件可以进行修改和回放。

1.录制和回放 录制:测试输入的自动化 回放:重新执行测试用例 2.检验 检验：设置检测点，对用例执行的正确性进行检查 3.可编程 编程：脚本更加灵活、功能更强大

单元测试与集成测试

单元测试

基本概念（软件单元、定义、意义、目标、实施者、关注点）

软件单元(Software Unit):一个应用程序中的最小可测部分

定义：对最小的软件设计单元（模块/源程序单元）的验证工作

意义： 1.消除软件单元本身的不确定性 2.其它测试阶段的必要的基础环节

目标： 1.单元体现了预期的功能 2.单元的运行能够覆盖预先设定的各种逻辑 3.单元工作中：内部数据能够保持完整性 4.可以接受正确数据，也能处理非法数据 5.在数据边界条件上，单元能正确工作 6.单元的算法合理，性能良好 7.扫描单元代码没有发现任何安全性问题

实施者：软件开发人员

关注点：1. 模块功能

2. 内部逻辑处理
3. 数据结构
4. 性能
5. 安全

单元测试规程（驱动器和程序桩）

单元测试通常与编码工作结合起来进行。

绝大多数情况下，单个单元是不能正常工作的，而是要和其他单元一起才能正常工作。

模块本身不是一个独立的程序，在测试模块时，必须为每个被测模块开发一个驱动器(driver) 和若干个桩程序(stub) 。

驱动器（Driver）： 对底层或子层模块进行测试时所编制的调用被测模块的程序，用以模拟被测模块的上级模块。

桩程序（Stub）： 对上层模块进行测试时，所编制的替代下层模块的程序，用以模拟被测模块工作过程中所调用的模块.

高内聚性程序: 驱动器和程序桩简单，错误容易被发现 低内聚性程序： 驱动器和程序桩工作量大。某些白盒测试需要推迟到集成测试阶段

手工和自动化测试 手工测试：按照需求规格设计测试用例完成测试；静态测试 自动测试：自动化方法的效果：有效验证较为独立单元的正确性

单元测试驱使程序员创建松耦合、高内聚的代码体，有助于开发健壮的软件

单元测试局限性 1.只验证单元自身的功能，不能捕获系统范围的错误 系统错误：集成错误、性能问题等 2.被测模块现实中可能接收的所有输入情况难以预料

集成测试

概念：把单独的软件模块结合在一起作为整体接受测试

接口：内部接口、外部接口；显示接口、隐含接口；

瞬时集成测试

当所有构件都通过单元测试，就把他们组合成一个最终系统，并观察它是否正常运转

缺陷： 1.无休止的错误：错误很多；错误修复很困难；修正错误后，新的错误马上出现 2.模块一次性结合，难以找出错误原因 3.接口错误和其它错误容易混淆

增量集成测试（自顶向下、自底向上）

将程序分成小的部分进行构造和测试

优点： 1.错误容易分离和修正 2.接口容易进行彻底测试

缺点： 会有额外开销，但能大大减少发现和修正错误的时间

自顶向下

首先集成主模块(主程序)，然后按照控制层次向下进行集成

深度优先、广度优先

优点： **1.尽早发现高层控制和决策错误 2.最多只需要一个驱动器 3.每步只增加一个模块** 4.支持深度优先和广度优先

缺点： 1.对低层模块行为的验证比较晚 2.需要编写额外程序（stub）模拟未测试的模块 3.部分测试用例由于依赖于其他层次的模块，在该模块未测试之前，这些测试用例的输入和输出很难确定

对于高层测试需要在低层测试完成后才可进行的情形难于编写桩模块

自底向上

原子模块->造件(Build)->应用软件系统

优点：

1. 尽早确认低层行为
2. **无需编写程序桩**
3. 对实现特定功能的树容易表示输入输出

缺点：

1. 推迟确认高层行为
2. **需编写驱动器**
3. 组合子树时，有许多元素要进行集成

测试插桩

两类插桩技术：黑盒插桩、白盒插桩

白盒插桩：语句覆盖插桩、分支覆盖插桩、条件覆盖插桩

白盒插桩作用：

（1）生成特定状态，检验状态的可达性；（2）显示或读取内部数据或私有数据；（3）监测不变数据（4）监测前提条件（5）人为触发事件时间（6）监测事件时间

系统测试、确认测试、回归测试

系统测试

概念

定义：

对完整集成后的产品和解决方案进行测试，用来评价系统对具体需求规格说明的功能和非功能的符合性的测试

意义：

1. 在测试中引入独立视角 独立测试团队；直接向高层报告
2. 在测试中引入客户视角 消除对产品的偏见；验证完整产品
3. 在测试中模拟客户使用环境 正式、完备和现实的测试环境
4. 测试产品功能和非功能的问题 产品交付给客户之前发现各种残余产品缺陷的最后机会。
5. 建立对产品的信心 把握系统测试发现缺陷的度。
6. 分析和降低产品发布的风险 对发现的缺陷进行分析和分类，修复高影响风险的缺陷。
7. 保证满足所有需求，产品具备交付确认测试条件

目的：

1.发现可能难以直接与模块或接口关联的缺陷 2.发现产品设计、体系和代码的基础问题（产品级缺陷）

实施者：

独立测试团队（引入独立视角）

功能测试

设计/体系结构测试

对照设计和体系结构开发和检查测试用例，从而整理出产品级测试用例。

业务垂直测试

针对不同业务纵深的产品，根据业务定制测试用例，验证业务运作和使用。

部署测试

验证系统是否能满足客户的部署需求

Alpha测试和Beta测试

α：用户在开发环境下进行的受控测试

β：用户在实际使用环境下进行测试；一种可以把待测产品交给客户收集反馈意见的机制

符合性的认证、标准和测试

产品需要通过主流硬件、操作系统、数据库和其他基础设施构件上进行的验证，并符合相关法规和行规。

非功能测试

非功能测试用于验证系统的质量因素。

可伸缩性测试/容量测试

确认产品参数的最大值，确定产品的主要瓶颈。

可靠性测试

在一定时间段内持续不断地测试软件产品，评价该产品在规定时间和条件下，维持其正常的功能操作、性能水平的程度。

压力测试

系统超过所描述的需求或资源限制时的情况；期望随着负载增加产品性能平稳下降

互操作性测试/兼容性测试

保证两个或多个产品可以交换和使用信息，并恰当地在一起运行

可使用性与易获得性测试

可使用性：以用户视点确认产品的易用性、速度和美感的测试

易获得性：检查产品对于行动不便的用户也可使用的测试

国际化测试

确保软件支持不同国家语言的测试手段

性能测试

为了获取或验证系统性能指标而进行的测试。

响应时间、吞吐率和系统的使用情况

安全性测试

确认测试

实施者：客户或客户代表

定义：检查产品是否满足在项目的需求阶段定义的确认准则，或者说是否具备在真实环境中使用的条件

引入时机：系统测试之后

目的：验证和接受产品

回归测试

概念：回归测试是对之前已测试过、经过修改的程序进行的重新测试，以保证该修改没有引入新的错误或者由于更改而发现之前未发现的错误。

组测试：一个模块可能单独工作正常，多个模块集成工作时却出现错误；组测试(Group Testing)是寻找此类错误的有效方法；

波及效应

波及效应是为了发现所有受影响部分和发现潜在的受影响部分，以保证软件发生改变后仍然保持一致性与完整性。