软件测试 Software Testing

1. 测试 Testing

1.1 测试基本概念 Testing - Basic Concepts

- **定义 Definition**：
  - 测试是软件验证和确认(V&V)过程的一部分
  - 从需求阶段就开始，贯穿整个开发过程
  - 目的：检查软件是否满足规格说明并提供预期功能
- **两种技术 Two Techniques**：
  1. 静态测试 Static Testing：
     - 关注静态系统表示分析
     - 包括文档和代码分析
  2. 动态测试 Dynamic Testing：
     - 使用测试数据执行程序
     - 观察运行行为

1.2 测试基础 Testing - The Basics

- **核心概念 Core Concepts**：
  - 使用人工数据执行程序
  - 检查测试运行结果是否存在错误和异常
  - 评估程序的非功能属性

1.3 测试目标 Testing - 2 Goals

1. **验证目标 Validation Goal**：
   - 向开发者和客户证明软件满足需求
   - 自定义软件：每个需求至少一个测试
   - 通用软件：测试所有系统特性及组合
2. **缺陷发现 Defect Discovery**：
   - 发现行为不正确或不符合规格的情况

- 关注：系统崩溃、不当交互、计算错误等

1.4 测试阶段 Stages of Testing

1. **开发测试 Development Testing**：
   - 开发过程中进行测试
   - 由系统设计者和程序员参与
2. **发布测试 Release Testing**：
   - 独立测试团队测试完整系统版本
   - 验证系统满足利益相关者需求
3. **用户测试 User Testing**：
   - 用户在其环境中测试系统
   - 包括验收测试

2. 开发测试 Development Testing

2.1 开发测试基础 Development Testing - The Basics

包含三个层次：

1. **单元测试 Unit Testing**：
   - 测试单个程序单元或对象类
   - 关注对象功能性
2. **组件测试 Component Testing**：
   - 测试多个集成单元
   - 关注单元间接口交互
3. **系统测试 System Testing**：
   - 测试完整系统
   - 关注组件交互

2.1.4 选择测试用例 Choosing A Test Case

定义和必要性 Definition & Necessity

- 定义：
  测试用例是为特定测试目标（如执行特定程序路径或验证是否符合特定需求）而设计的一组测试输入、执行条件和预期结果。
- 必要性：
  1. 测试资源有限：
     - 时间和成本限制
     - 无法进行穷尽测试
  2. 测试效率：
     - 需要用最少的测试用例发现最多的缺陷
     - 确保关键功能正确性
  3. 质量保证：
     - 系统可靠性验证
     - 降低发布风险

测试用例类型 Test Case Types

1. 正常操作测试 Normal Operation Tests
2. - 定义：验证在预期输入和正常条件下系统的行为
   - 好处：
     1. 确保核心功能正常工作
     2. 建立基准行为标准
     3. 验证用户常见使用场景
3. 异常情况测试 Abnormal Input Tests
4. - 定义：使用边界值、无效数据和异常情况测试系统
   - 好处：
     1. 发现边界情况bug
     2. 验证错误处理机制
     3. 提高系统稳定性

2.1.5 分区策略 Partition Strategy

定义和必要性 Definition & Necessity

- 定义：
  将输入或输出数据划分为有限个等价类别（分区），认为每个分区中的数据对于测试目的具有相同的行为。
- 必要性：
  1. 降低测试复杂度：
     - 将无限的输入空间简化为有限的分区
     - 使测试更加可管理
  2. 提高测试效率：
     - 避免冗余测试
     - 确保覆盖所有重要情况
  3. 系统化测试方法：
     - 提供结构化的测试设计方法
     - 减少随机性和主观性

实施策略 Implementation Strategy

1. 识别分区：
   - 根据规格说明
   - 基于有效/无效条件
   - 考虑边界情况
2. 选择测试用例：
   - 分区边界值
   - 分区中间值
   - 特殊值
3. 设计测试套件：
   - 确保每个分区至少一个测试
   - 优先测试边界条件
   - 考虑分区组合

优势 Benefits

1. 测试覆盖优化：

- 减少测试用例数量
- 保持测试效果

2. 缺陷发现效率：
   - 边界值测试效果好
   - 容易发现分区间的问题
3. 维护性：
   - 测试结构清晰
   - 易于更新和扩展

2.1.6 指南策略 Guideline Strategy

定义和必要性 Definition & Necessity

- 定义：
  基于过往测试经验和最佳实践制定的测试指导原则，用于设计和选择测试用例。
- 必要性：
  1. 经验传承：
     - 利用历史测试经验
     - 避免重复错误
  2. 标准化：
     - 提供一致的测试方法
     - 确保测试质量
  3. 效率提升：
     - 减少测试设计时间
     - 提高测试有效性

具体指南及其价值 Guidelines and Values

1. 单值序列测试 Single-Value Sequence
   - 定义：测试只包含一个元素的序列
   - 价值：
     1. 发现特殊情况处理问题
     2. 验证边界条件

3. 检查最小输入处理

2. 不同大小序列测试 Various-Size Sequences
   - 定义：使用不同长度的序列进行测试
   - 价值：
     1. 验证规模可扩展性
     2. 检查内存管理
     3. 发现性能瓶颈
3. 序列元素访问测试 Sequence Access Testing
   - 定义：测试序列中不同位置元素的访问
   - 价值：
     1. 验证索引处理
     2. 检查边界条件
     3. 确保一致性处理
4. 空序列测试 Empty Sequence
   - 定义：测试空序列或零长度输入
   - 价值：
     1. 验证边界情况处理
     2. 检查错误处理
     3. 防止空指针异常

实施建议 Implementation Recommendations

1. 系统化应用：
   - 创建测试清单
   - 确保指南覆盖
2. 持续改进：
   - 根据新发现更新指南
   - 添加特定领域规则
3. 团队协作：
   - 共享测试经验

- 统一测试标准

2.2 单元测试 Unit Testing

详细说明：

- 以对象类为测试单位
- 完整测试覆盖包括：
- 1. 测试所有对象相关操作
  2. 设置和查询所有对象属性
  3. 测试所有可能状态

2.2.1 单元测试示例 - Weather Station

2.2.2 自动化单元测试 Automated Unit Testing

测试框架结构：

1. 设置部分 Setup：
   - 初始化测试对象
   - 准备测试数据
2. 调用部分 Call：
   - 执行被测试方法
3. 断言部分 Assert：
   - 验证测试结果

2.3 组件测试 Component Testing

- 定义：
  组件测试是对由多个交互单元组成的复合组件进行测试，重点关注组件间的接口交互和集成功能。
- 必要性：
  1. 验证集成：
     - 确保单元间正确交互

- 验证接口兼容性

2. 发现集成缺陷：
   - 单元测试无法发现的接口问题
   - 组件间通信问题
3. 系统稳定性：
   - 确保组件集成的可靠性
   - 验证复合功能的正确性

接口错误类型：

1. 接口误用 Interface Misuse
2. 接口误解 Interface Misunderstanding
3. 时序错误 Timing Errors

测试指南：

1. 测试参数极限值
2. 测试空指针参数
3. 进行压力测试
4. 测试共享内存访问顺序

2.4 系统测试 System Testing

定义和必要性 Definition & Necessity

- 定义：
  系统测试是在开发过程中集成组件创建系统版本并进行测试的过程，重点关注组件间的交互。
- 必要性：
  1. 集成验证：
     - 确保所有组件能协同工作
     - 验证系统级功能
  2. 交互检查：
     - 测试组件间通信
     - 验证数据传输时机和准确性

3. 整体功能：
    - 验证系统层面的需求
    - 确保端到端功能正确

系统测试 vs 组件测试 System vs Component Testing

- 定义：
  系统测试是在开发过程中集成组件创建系统版本并进行测试的过程，重点关注组件间的交互。
- 必要性：
  1. 集成验证：
     - 确保所有组件能协同工作
     - 验证系统级功能
  2. 交互检查：
     - 测试组件间通信
     - 验证数据传输时机和准确性
  3. 整体功能：
     - 验证系统层面的需求
     - 确保端到端功能正确

2.4.1 基于用例的测试方法 Use Case-based Approach

方法定义和价值 Definition and Value

- 定义：
  使用系统用例作为测试基础，验证系统交互和操作流程。
- 应用：
  1. 真实场景测试
  2. 覆盖用户交互
  3. 验证业务流程

实施方法 Implementation

1. 用例分析：
    - 识别关键用例

- 定义测试范围
- 确定测试优先级

2. 测试设计：
   - 基于序列图设计测试
   - 识别测试操作
   - 创建测试场景
3. 执行验证：
   - 按用例流程测试
   - 验证预期结果
   - 记录测试结果

2.4.2 测试用例设计 Test Case Design

1. 用例场景示例：
   - Start > Checking > SeatUnavailable > Terminate

- Start > Checking > Registering > Registered > Terminate
- Start > Checking > Registering > SeatUnavailable > Terminate

2. 设计考虑：
   - 覆盖主要流程
   - 包含异常情况
   - 验证状态转换

2.4.3 系统测试策略 System Testing Policies

测试策略定义 Testing Policy Definition

- 定义：
  确定系统测试覆盖范围的政策和规则。
- 重要性：
  1. 确保测试完整性
  2. 标准化测试过程
  3. 提高测试效率

测试政策示例 Example Testing Policies

. 功能测试要求：

- 测试所有菜单访问的功能
- 示例：文件菜单下的打开、保存、退出功能

2. 组合功能测试：
   - 测试通过相同菜单访问的功能组合
   - 示例：文本格式化组合（粗体+斜体）
3. 输入验证测试：
   - 使用正确和错误的输入测试所有功能
   - 示例：用户名密码验证

实施指南 Implementation Guidelines

1. 测试计划制定：

- 定义测试范围
- 设置优先级
- 分配资源

2. 测试执行：
   - 按政策要求执行
   - 记录测试结果
   - 追踪问题修复
3. 测试评估：
   - 评估测试覆盖
   - 分析测试效果
   - 持续改进策略

实际用例 Best Practices

1. 测试自动化：
   - 自动化重复测试
   - 回归测试自动化
   - 持续集成测试
2. 文档管理：
   - 维护测试文档
   - 更新测试用例
   - 记录测试结果
3. 质量监控：
   - 跟踪缺陷趋势
   - 评估测试效果
   - 优化测试策略

3. 发布测试 Release Testing

3.1 发布测试基本概念 Release Testing Basics

- 定义 Definition：

- 测试系统发布版本，供开发团队外部使用
- 主要目标：说服供应商系统适合使用
- 验证功能、性能和可靠性

3.2 发布测试与系统测试的区别 Release Testing vs System Testing

- 关键差异：
  1. 执行团队：
     - 由独立测试团队负责
     - 测试团队未参与系统开发
  2. 测试目标：
     - 系统测试：发现系统缺陷
     - 发布测试：验证系统满足需求，适合外部使用

3.3 基于需求的测试方法 Requirements Based Approach

示例：

1. 需求描述：
   - 病人对药物过敏时，开具处方需显示警告
   - 忽略过敏警告时，需提供原因
2. 测试设置：
   - 设置无已知过敏的病人记录
   - 设置有已知过敏的病人记录
   - 设置对多种药物过敏的记录
   - 测试开具过敏药物处方
   - 测试警告覆盖情况

3.4 性能测试 Testing Performance

1. 性能测试计划：
   - 逐步增加负载
   - 直到性能不可接受
   - 监控系统响应

2. 操作概要 Operational Profile：
3. 示例场景：
    - 90% A类型交易
    - 5% B类型交易
    - 剩余C、D、E类型
      根据实际使用比例设计测试
4. 压力测试 Stress Testing：
    - 超出设计限制的测试
    - 目的：
        - 测试系统失败行为
        - 发现常规测试发现不了的缺陷

4. 用户测试 User Testing

4.1 用户测试基础

- 重要性：
    - 即使完成系统和发布测试也必要
    - 真实环境影响系统可靠性和性能
    - 无法在测试环境完全复制

4.2 Alpha测试 Alpha Testing

- 特点：
    - 用户与开发团队共同测试
    - 在开发者场地进行
    - 早期发现问题和改进机会
- 优势：
    1. 早期发现问题：
        - 识别细微和环境相关问题
        - 避免后期高成本修复
    2. 早期使用者参与：

        - 增强用户对产品的主人翁感
        - 有助于产品上市后的支持

4.3 Beta测试 Beta Testing

- 定义：
    - 向用户发布早期版本
    - 供评估使用
    - 发现环境交互问题
- 优势：
    1. 可用性反馈：
        - 了解真实用户交互
        - 发现可用性问题
    2. 性能问题：
        - 识别性能瓶颈
        - 发现扩展性问题
    3. 市场洞察：
        - 收集目标市场反馈
        - 确保产品满足期望

4.4 验收测试 Acceptance Testing

4.4.1-4.4.6 验收测试流程 Acceptance Testing Process

1. 定义验收标准 Define Acceptance Criteria：
    - 在合同签订前确定
    - 可能随开发过程调整
2. 规划验收测试 Plan Acceptance Testing：

    考虑因素：
    - 需求覆盖程度
    - 测试特性顺序
    - 风险评估和缓解

3.

4. 设计验收测试 Derive Acceptance Tests：
    - 测试功能特性
    - 测试非功能特性
    - 基于验收标准设计
5. 执行验收测试 Run Acceptance Tests：
    - 理想情况：在实际使用环境
    - 实际情况：可能需要模拟环境
6. 协商测试结果 Negotiate Test Results：
    - 讨论测试未通过项
    - 确定开发团队响应方案
7. 系统接受/拒绝决定 Reject/Accept System：
    - 基于测试结果做出决定
    - 如不满足要求，需进一步开发
    - 完成后重复验收测试