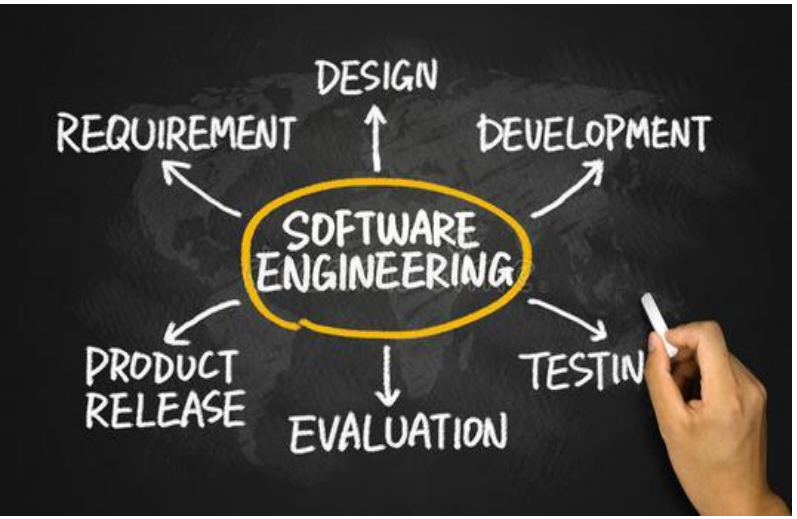
软件工程基础 —— 第19章 测试面向对象的应用系统



计算机学院 孟宇龙

目录

关键概念

- 类测试
- 簇测试
- 一致性
- 基于故障的测试
- 多类测试
- 面向对象模型
- 划分测试
- 随机测试
- 基于场景的测试
- 测试方法
- 测试策略
- 基于线程的测试
- 基于使用的测试

OO 测试

- 为了充分测试面向对象的系统，必须做3件事：
  - 对测试的定义进行扩展，使其包括**应用于面向对象分析和设计模型的错误发现技术**
  - 单元测试和集成测试策略必须彻底改变
  - 测试用例设计必须考虑面向对象软件的独特性质

OO 测试

- 尽力发现当类之间存在协作以及子系统穿越体系结构层通信时可能发生的错误
- 在策略上与传统测试相似，在战术上不同

19.1 测试OO 模型

- 面向对象分析和设计模型的评审非常有用，因为相同的语义结构（例如，类、属性、操作、消息）出现在分析、设计和代码层次。
- 因此，在分析期间所发现的类属性的定义问题会防止副作用的发生。如果问题直到设计或编码阶段（或者是分析的下一轮迭代）还没有出现，副作用就会发生。

19.2.1 OO 模型的正确性

- 在分析和设计期间，可以根据模型是否符合真实世界的问题域来评估模型的语义正确性。
- 如果模型准确地反应了真实世界（详细程度与模型被评审的开发阶段相适应），则在语义上是正确的。
- 实际上，为了确定模型是否反应了真实世界的需求，应该将其介绍给问题领域的专家，由专家检查类定义和层次中遗漏和不清楚的地方。
- 要对类关系（实例连接）进行评估，确定这些关系是否准确地反应了真实世界的对象连接。

19.2.2 面向对象模型的一致性

- 为了评估的一致性，应该检查每个类及与其他类的连接。可以使用**类—责任—协作模型**或对象—关系图来辅助此活动。协作意味着面向对象系统的类之间的一系列关系。对象关系模型提供了类之间连接的图形表示。这些信息可以从分析模型中获得。

19.2.2 面向对象模型的一致性

推荐使用下面的步骤对类模型进行评估：

1. 检查CRC模型和对象-关系模型。
2. 检查每一张CRC索引卡片的描述以确定委托责任是定义协作者的一部分。
3. 反转连接，确保每个提供服务的协作者都从合理的地方收到请求。
4. 使用前一步骤中反转后的连接，确定是否真正需要其他类，或者责任类之间的组织是否合适。
5. 确定是否可以将广泛请求的多个责任组合为一个责任。

19.3 OO 测试策略

- 面向对象软件的测试不同于传统的软件测试，其策略和概念发生了下述变化：
- 单元测试
  - 单元的概念发生了改变
  - 最小的测试单元是封装了的类
  - 已经不可能再独立地测试单一的操作了（独立地测试单一的操作是单元测试的传统观点），而是作为类的一部分进行测试

19.3 OO 测试策略

- 集成测试
  - 基于线程的测试，将响应系统的一个输入或事件所需的一组类集成到一起
  - 基于使用的测试，通过测试很少使用服务类（如果有的话）的那些类（称之为独立类）开始构造系统
  - 簇测试[McG94] 通过设计试图发现协作错误的测试用例，对一簇协作类（通过检查CRC和对象-关系模型来确定）进行测试

19.3 OO 测试策略

- 确认测试
  - 类连接的细节消失了
  - 利用用例，它是需求模型的一部分
  - 传统的黑盒测试方法可以用于驱动确认测试

19.4 OO测试方法

面向对象软件的测试用例设计方法还在不断改进，然而，
Berard 已经提出了以下总结方法：

1. 每个测试用例都应该唯一地标识，并明确地与被测试的类相关联
2. 应该叙述测试的目的
3. 应该为每个测试开发测试步骤，并包含以下内容[BER94]：
   a. 将要测试的类的指定状态列表
   b. 作为测试结果要进行检查的消息和操作列表
   c. 对类进行测试时可能 发生异常列表
   d. 外部条件列表（即软件外部环境的变更，为了正确地进行测试，这种环境必须存在）
   e. 有助于理解或实现测试的补充信息

测试方法

- **基于故障的测试**
  - 测试人员查找似然故障（即系统的实现有可能产生错误的方面）。为了确定这些故障是否存在，需要设计测试用例以检查设计或代码
- **类测试与类层次**
  - 继承并不能排除对所有派生类进行全面测试的需要。事实上，它确实使测试过程更复杂。
- **基于场景的测试设计**
  - 基于场景的测试关心用户做什么，而不是产品做什么。这意味着捕获用户必须完成的任务（通过用例），然后在测试时使用它们及其变体。

19.5.1 OOT 方法：随机测试

- 随机测试
  - 确定适用于类的操作
  - 定义约束它们的使用
  - 确定最小测试序列
    - 操作序列定义了类的最小生命历时
  - 随机产生（但是有效）一些不同的测试序列
    - 检查其他类实例（更复杂的）的生命历时

19.5.2 OOT 方法：划分测试

- 划分测试
  - 与传统软件的等价划分基本相似，划分测试减少测试特定类所需的测试用例数量
  - 基于状态划分
    - 根据它们改变类的状态的能力对类操作进行分类
  - 基于属性划分
    - 根据它们所使用的属性对类操作进行分类
  - 基于类别划分
    - 根据每个操作所完成的一般功能对类操作进行分类

19.6 OOT 方法：类间测试

- 与单个类的测试相类似，类协作测试可以通过运用随机和划分方法、基于场景测试及行为测试来完成。下面的方法是生成多类随机测试用例的方法：
- 类间测试
  - 对每个客户类，使用类操作列表来生成一系列随机测试序列。这些操作将向其他服务类发送消息。
  - 对生成的每个消息，确定协作类和服务对象中的相应操作。
  - 对服务对象中的每个操作（已被来自客户对象的消息停用），确定它传送的消息。
  - 对每个消息，确定下一层被调用的操作，并将其引入到测试序列中。

196.2 OOT 方法：行为测试

类的状态图可用于辅助生成检查类的动态行为的测试序列。右图给出了前面讨论的Account类的状态图。

将要设计的测试应该覆盖所有的状态，也就是说，操作序列应该使Account类能够向所有可允许的状态转换