测试是最好的设计

付仲恺
微软特邀开发专家

议题

- 软件开发方法论
  - 敏捷软件开发 vs 传统开发方式
- 什么是TDD
- 设计可测试应用程序
  - 领域模型应用程序设计模式
- 使用TDD改进设计
- NUnit&NMock介绍

软件开发方法论
传统开发方式

- 软件就是将抽象的工件，如：业务逻辑需求，精炼为具体的工件，如：代码，的有系统的产品。
- 对于每一个工件，无论其处于哪个级别，都被机械地看作是系统的一部分，并且因此必须与系统的其他工件保持协调

软件开发方法论
敏捷开发方式

- 软件是经过精炼的代码和测试的有系统的产品
- 工件是用于与团队其他成员交流意见的有用的工具，而不是最终结果

什么是敏捷开发方式

- 敏捷软件开发宣言
  - 个体和交互　胜过　过程和工具
  - 可以工作的软件　胜过　面面俱到的文档
  - 客户合作　胜过　合同谈判
  - 响应变化　胜过　遵循计划
- 极限编程是敏捷软件开发方式的一种
  - TDD是极限编程的重要组成部分之一

什么是TDD

测试驱动开发是一种开发风格:

- 在写产品代码之前，先写它的单元测试
- 没有单元测试的类不允许作为产品代码
- 单元测试例子决定了如何写产品代码
- 不断地成功运行所有的单元测试例子
- 不断的完善单元测试例子
- 避免设计过度

TDD的基本观点

- 在TDD中，对于应用程序的绝大多数部分都可以采用如下迭代步骤进行开发：
  - 开发前先编写测试用例
  - 如果测试用例在开发的代码上失败
    - 修改错误
    - 重复测试
  - 进行下一步测试开发
- 对于每个类进行分别测试
- 对于每部分进行简单设计，频繁重构

测试先行

- 在写任何代码之前，先编写测试用例
- 测试先行是：
  - 面向对象的分析，设计和开发方法
  - 质量控制方法
  - 重构和优化的方法

TDD对开发者和管理者的意义

- 对开发者来说，通过运行单元测试以及每日构建，每天都可以清楚的知道自己的代码是能够正常工作的。
- 对管理者来说，通过每日构建的结果，每天都清楚的知道项目的质量和开发进度。

什么时候编写测试用例

- 在编写新功能和方法之前，先要编写测试用例
- 如果要在没有经过测试的代码上写新的功能，请先写目前代码的测试用例
- 如果要修正一个Bug，请先为这个Bug写一个测试用例
- 如果要重构没有测试过的代码，请先写一个测试用例

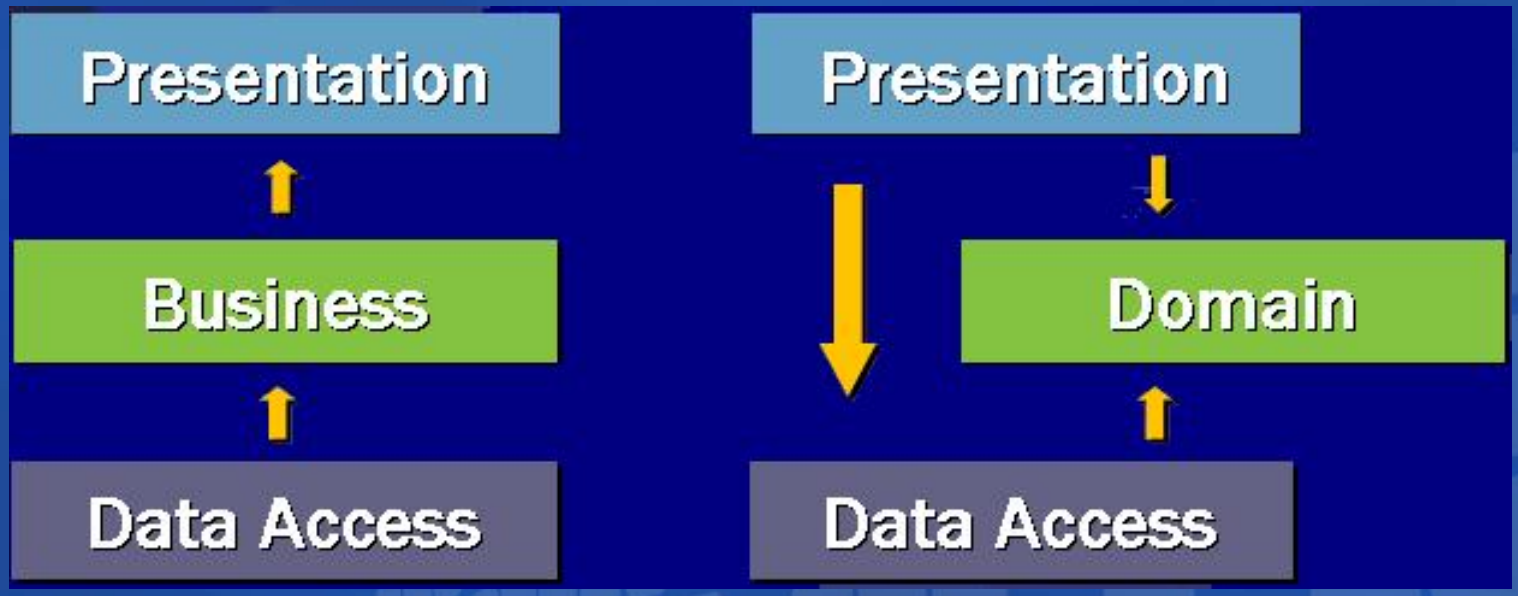
需要编写哪些测试用例

- 结果是否正确
  - 数值是否正确
  - 结果集合顺序是否正确
- 所有的边界条件是否正确
  - 0和非0测试
  - 加减1测试
- 用反向逻辑来验证结果
- 使用其它方法反复验证结果
- 强迫异常条件发生
- 性能是否在允许的范围内

设计"可测试"应用程序
领域模型模式

- 领域类在代码中表示了业务逻辑
- 设计原则：依赖倒置原则(DIP)
  - 抽象不应该依赖于细节
  - 细节应该依赖于抽象
- 领域类不依赖于任何形式的数据访问技术
- 领域类不依赖于任何形式的用户交互技术
- 对于某些具体问题(例如:安全),当在领域类中描述时,不应该影响模型的业务逻辑的能力

设计“可测试”应用程序
DNA vs 领域模型



- 箭头方向表示程序集依赖关系

使用TDD改进设计
从可测试系统设计中获益

- 降低耦合度
  - 业务逻辑独立于领域层
  - 应用程序的所有其它层次采用领域层语言而不是平台指定语法进行通讯
- 简化了分布式系统的构建
  - 在远程应用环境中,领域对象能够以MarshalByValue对象的方式传输或者添加XML序列化属性符号,以用于Web Services
  - 数据访问对象成为服务对象
- 降低了大量的复杂的,不必要的代码.

使用TDD改进设计
测试接口

- 首先编写测试用例可以按照类的使用者的观点来进行类的设计
- 为接口编写测试用例事实上暗示着随着时间的推移对于类设计的改变

使用TDD改进设计
改善组件对象模型

- 采用组件继承的对象模型变得更加灵活和便于维护
- 使用TDD可以生成具有独立测试能力,以及能够处理更加复杂行为的具有高度规范化的对象
- 设计可测试领域类揭示了能够抽取厂类或者构建类的逻辑

NUnit介绍
功能描述

- 开源测试框架
  - http://sourceforge.net/projects/nunit
  - http://www.nunit.org/
- 拥有两个基本框架
  - 用于运行测试的GUI界面
  - 用于定义测试的类库
- 通过属性标记的方式来实现测试设计
- 能够测试任何编译成CRL的.NET语言所开发的应用程序

为什么使用NUnit

- NUnit允许开发者在快速编写代码的同时仍然保证代码质量
  - 减少代码调试时间
  - 增强对自身编写代码正确工作的信心
  - 编写测试
    - 对于代码重构和增加新特性更加有信心
  - 不编写测试
    - 由于不知道结果，使得代码重构和新特性增加成为了妄想
- NUnit非常简单
  - 使用NUnit，开发者能够快速编写执行你的代码的测试，并且随着软件的增长编写新的测试用例

为什么使用NUnit

- NUnit测试用例能够检查自身结果并且立即提供反馈
  - 测试能够自动运行，并且自动检查测试结果
- 编写NUnit测试用例非常简单
  - 使用NUnit测试框架，开发者能够非常容易地编写测试用例
- NUnit测试能够增加软件的稳定性
  - 通过测试形成了软件结构完整性的整合

NUnit测试划分

- 将应用程序看作：
  - 带有一组或者多组TestFixture属性的应用程序
- 将TestFixture分在单一测试用例中
  - 对于对象的每一个操作都创建一个测试
  - 修正所有操作然后继续测试

NUnit测试框架

- 必须标记的属性
  - [TestFixture]
    - 用于表示类中包含测试方法
  - [Test]
    - 用于表示该方法应该被NUnit调用

NUnit测试框架

- 可选属性
  - [TestFixtureSetUp]
    - 用于表示启动方法。标记了该属性的方法会在测试启动前被调用
  - [TestFixtureTearDown]
    - 用于表示结束方法。标记了该属性的方法会在所有测试方法都运行结束后被调用

NUnit测试框架

- [SetUp]
  - 标记了该属性的方法，会在每个测试运行前被调用
- [TearDown]
  - 标记了该属性的方法，会在每个测试运行后被调用

NUnit测试框架
测试流程顺序

- 测试开始
  - 调用TestFixtureSetUp
    - 调用SetUp
      - 调用ExampleTestOne
    - 调用TearDown
    - 调用SetUp
      - 调用ExampleTestTwo
    - 调用TearDown
  - 调用TestFixtureTearDown
- 测试结束

NUnit测试框架

- [ExpectedException(typeof(Exception))]
  - 当希望抛出异常时
  - 只有当抛出异常时测试才能通过
- [Ignore(“Not ready for primetime")]
  - 当测试没有准备好时，标记该属性表示不要运行该测试
- Category("GroupName")
  - 对测试进行分组处理
  - 与[Test]以及[TestFixture]属性共同使用

演示

NUnit使用演示

NMock介绍

- 单元测试对象模拟工具
  - 对于一些对其他类或者系统组件有依赖而难以测试的组件，可以使用对象模拟技术
  - 模拟对象方法允许用模拟的版本来代替被依赖的类
  - NMock是一个组件，需要在NUnit环境中使用
  - NMock通过C#反射机制，实现对象的动态模拟

NMock介绍

- 在以下情况中，通常使用NMock来模拟对象：
  - 具有不确定行为的真实的对象
  - 真实对象非常难以创建
  - 真实对象拥有难以出发的行为
  - 真实对象非常缓慢
  - 真实对象拥有或者本身就是用户界面
  - 测试需要知道真实对象如何被使用
  - 真实对象不存在

NMock操作步骤

- 首先实例化一个 Mock 对象
  - 构建时要将要模拟的接口或类的类型传递给 Mock 对象作为构建参数
- 记录 Mock 对象的行为
- 通过 Mock 对象的属性来获得一个模拟类型的示例

演示

NMock使用演示

总结

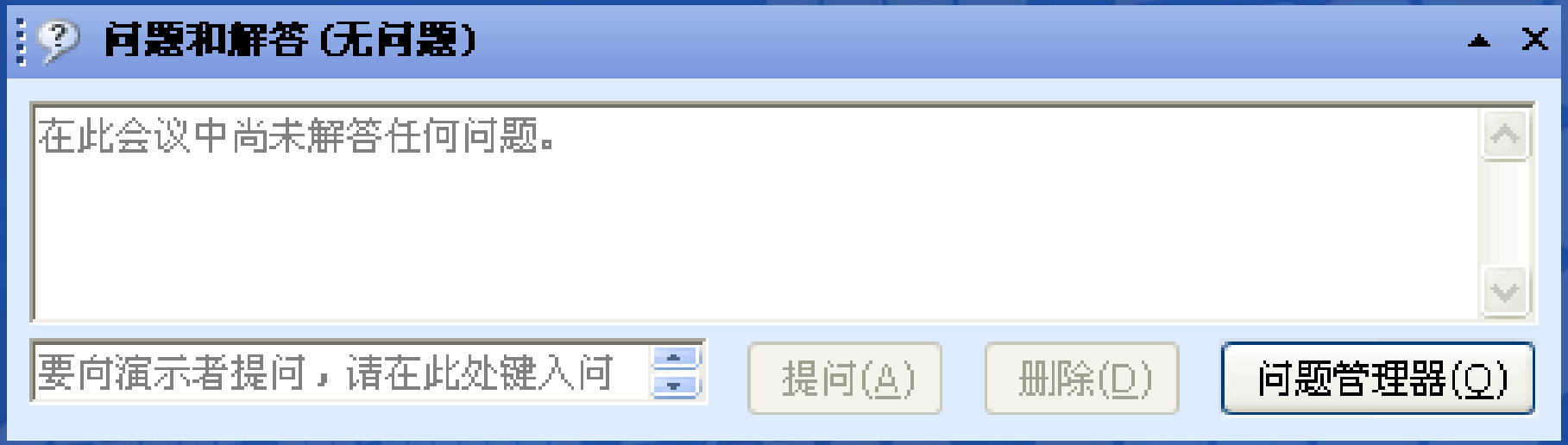
- 敏捷软件开发
- 测试驱动开发原理
- 构建可测试应用程序
  - 领域模型模式
- NUnit & NMock

您的潜力，我们的动力

Microsoft
微软(中国)有限公司

Q&A

如需提出问题，请单击“提问”按钮并在随后显示的浮动面板中输入问题内容。一旦完成问题输入后，请单击“提问”按钮。




您的潜力，我们的动力
Microsoft
微软（中国）有限公司
Microsoft
msdn
MSDN Webcasts