

# Taking TDD to the Next Level

**Erik Doernenburg**  
**Software Developer**  
**ThoughtWorks, Inc.**

# Recap

- **TDD isn't about testing – it's about programming!**
- **The red-green-refactor mantra:**
  - **write test, write code, refactor (repeat)**
- **State verification**
  - **setup objects, invoke functionality, assert state**
- **Behaviour verification**
  - **replace neighbours with mocks, verify interactions**



*switch to IDE and show a simple state-based test (order total) and a test with mocks (cart and inventory)*

# Tight coupling is bad (doh!)

- **Creating or referencing concrete implementations is problematic:**
  - **Hard to re-use service**
  - **Hard to extend**
- **Really bad for testability**
  - **Mocks only work if we can substitute collaborators**
  - **Without mocks, where does the test data come from?**

**Interface/impl separation improves testability!**

# Why Dependency Injection?

- **If we want to substitute the collaborators, they must be provided from 'outside'**
- **With Dependency Injection dependent components are injected from the outside**
- **Components are not concerned with creating dependent components**
- **Dependency Injection is a a natural fit**



***switch to IDE and show how service locators make previous example awkward***

# The return of the stub?

- **Dynamic proxies evolved from stub objects**
- **Sometimes an interaction is complex and it is hard to use dynamic mocks**
- **Option 1: Introduce a stub object to record and assert state later**
- **Option 2: Use composition and avoid issue**

**Better testability = Better design**



*switch to IDE, starting from the problem (order message), show implementation with mock, show implementation with stub, then refactor (extract message creator)*

# Test Doubles

- **Dummy**
  - **Passed around, never really used**
- **Fake**
  - **Have working implementation**
- **Stub**
  - **Provide canned answers and/or recording**
- **Mock**
  - **Verify pre-programmed expectations**

# How do I test internal methods?

- **Make them available!**
  - **Make them public on implementation but do not add to interface**
- **Subclass with inner class in test!**
  - **Doesn't work when things are private or when we can't substitute objects**
- **Decompose!**
  - **But don't end up writing global functions**

**Better testability = Better design**



***switch to IDE, show test of method  
(sendOrderMessage) as public method  
and with subclass in test; then  
compare to decomposed version  
(message creator)***

## How do I test *this*?!

- **Sometimes a small bit of code is in the way, no matter where we move it.**
- **Remember: We're testing to make our life easier, not to achieve 100% coverage!**
- **Isolate that code as much as possible and don't write a unit test for it.**
- **We have automated acceptance tests, right?**

**Be pragmatic!**

*switch to IDE, show how the code that  
reads the excel sheet makes testing  
hard, in service as well as in controller;  
then introduce the solution: a tiny  
untested method*

# Object Mother

- **Combining DDD and TDD, we can write a lot of code without thinking about persistence**
- **Use an 'Object Mother' to create domain objects for the tests**
- **This is also *the* place to use reflection to set values on immutable objects**



***switch to IDE, show how most of the test method is object setup, which has re-use potential, move this into an object mother***

# Reference

- Mocks and Stubs essay  
**[martinfowler.com/articles/mocksArentStubs.html](http://martinfowler.com/articles/mocksArentStubs.html)**
- Test Double patterns  
**[xunitpatterns.com/Test%20Double%20Patterns.html](http://xunitpatterns.com/Test%20Double%20Patterns.html)**
- Object Mother pattern  
**[www.xpuniverse.com/2001/pdfs/Testing03.pdf](http://www.xpuniverse.com/2001/pdfs/Testing03.pdf)**

[erik.doernenburg.com](http://erik.doernenburg.com)