

Taking TDD to the Next Level

**Erik Doernenburg
Principal Consultant
ThoughtWorks, Inc.**

Recap

TDD isn't about testing

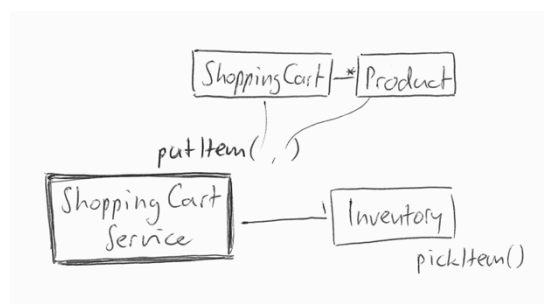
The red-green-refactor mantra

State verification

Behaviour verification

Tools have changed

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



Why “new” is evil

```
public class FooService {  
    private BarDataSource bar;  
  
    public Foo() {  
        bar = new Bar();  
    }  
  
    public doSomething() {  
        bar.whatever();  
    }  
}
```

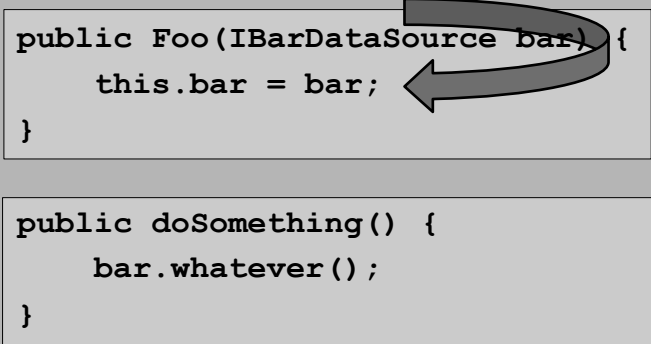


The role of Interface/Impl separation

```
public class FooService {  
    private IBarDataSource bar;  
  
    public Foo() {  
        bar = new ;  
    }  
  
    public doSomething() {  
        bar.whatever();  
    }  
}
```

Dependency Injection

```
public class FooService {  
    private IBarDataSource bar;  
  
    public Foo(IBarDataSource bar) {  
        this.bar = bar;  
    }  
  
    public doSomething() {  
        bar.whatever();  
    }  
}
```



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

The return of the stub?

Dynamic mocks evolved from stub objects

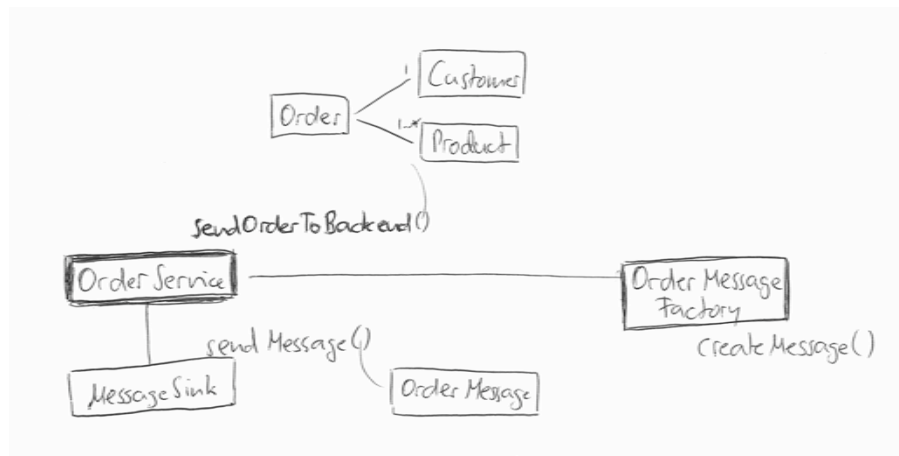
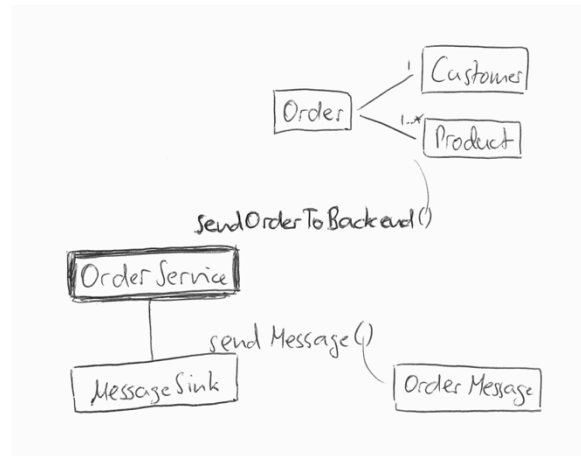
Sometimes dynamic mocks don't work out

Option 1: Use stub to turn behaviour into state

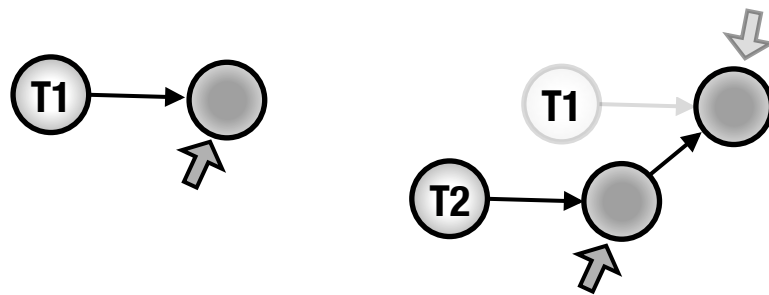
Option 2: Use composition and avoid issue

Better testability = Better design

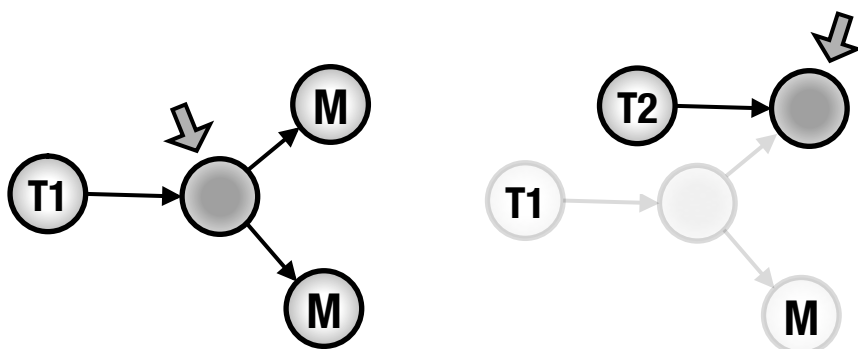
switch to IDE, starting from the problem (order message), show implementation with mock, show implementation with stub, then refactor (extract message factory)



Inside-out testing – the Classicists



Outside-in testing – the Mockists



How do I test *this*?!

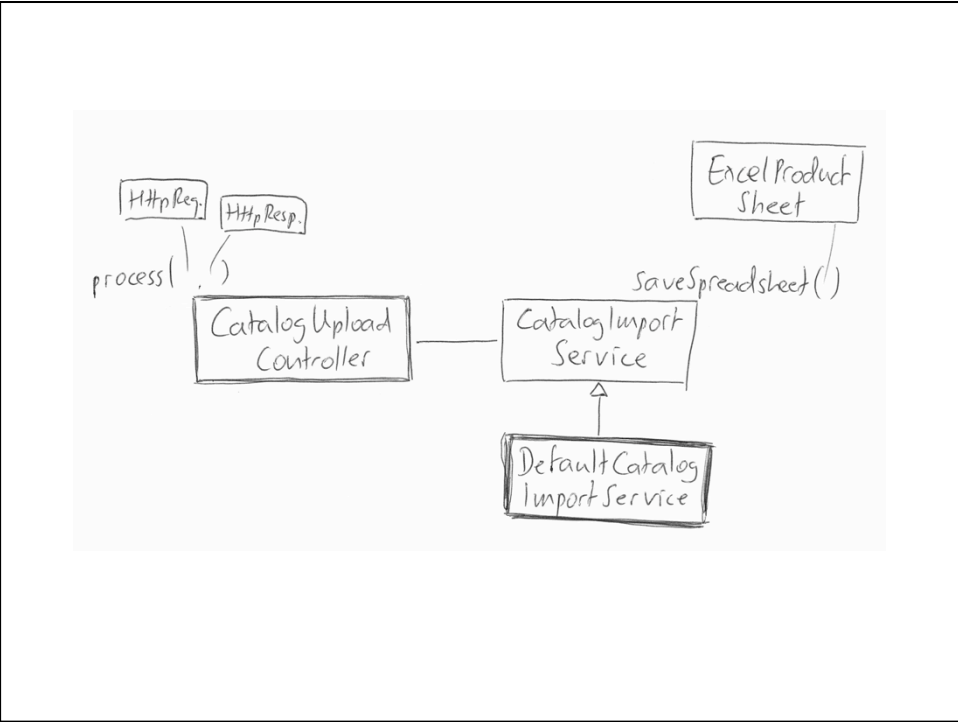
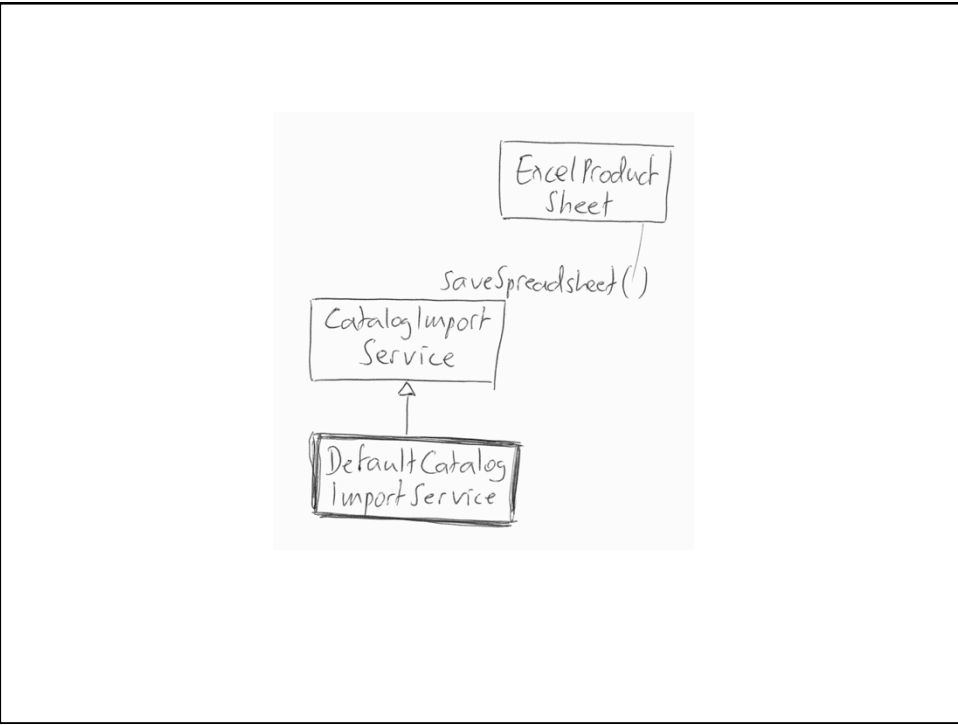
Sometimes a small bit of code is in the way...

Step 1: Isolate that code as much as possible

Step 2: Rely on the acceptance tests

*We're testing to make our life easier,
not to achieve 100% coverage!*

*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*



Test Doubles

Mock Verify pre-programmed expectations

Stub Provide canned answers and/or recording

Dummy Passed around, never really used

Fake Have working implementation

How do I test internal methods?

- a) Make them available
- b) Subclass with inner class in test
- c) Decompose

Better testability = Better design

*switch to IDE, show test of method
(readProduct) as public method and with
subclass in test*

Questions?

erik.doernenburg.com/downloads/tdd2.zip
www.jaoo.com.au
www.thoughtworks.com

Reference

Mocks and Stubs essay

martinfowler.com/articles/mocksArentStubs.html

Test Double patterns

xunitpatterns.com/Test%20Double%20Patterns.html

Hamcrest

code.google.com/p/hamcrest

Mockito

mockito.org