

Mocks Suck



**CODE**Dexterity



2000

The year it began. Steve Freeman, Tim Mackinnon & Philip Craig unleash the term 'Mock' in the paper “Endo Testing: Unit testing with mock objects” at the XP2000 conference. Reaction was mixed.

# MockObjects™

A TDD variant that emphasises a particular design style. The "no getters rule", results in heavy use of double dispatch/visitor and lots of interfaces (in statically typed languages).

## mock objects

A specific type of test double where you set expectations on what interactions it will receive and that verifies that those expectations were met. Can be hand coded but more usually use associated frameworks that allow you to specify the expectations and have them checked automatically.

## Mocks aren't Stubs

Because the term 'mock' is catchy it becomes widely adopted but leads to confusion as people deviate from the 'true' meaning of mocks. Mocks aren't stubs is the title of an article by Martin Fowler that tries to clarify the differences.

## No Getters and MVC

MVC frameworks like Rails uses lots of "getters", predominantly in the view, so it's difficult to apply the MockObjects design style to application using frameworks. The sample application in the book, "Growing Object-Oriented Software, Guided by Tests" is a Java Swing app built from scratch for example.

## Thick Glue

The MockObjects design style tends towards lots of interfaces (in statically typed languages) and premature abstraction, making it easy to lose sight of the big picture. Lots of interactions but nothing working. The paper “Using Mocks and Tests to Design Role-Based Objects” is a good example of over abstraction in a point of sale application.

# Dependency Injection

Passing dependencies as parameters is a design style that follows from TDD, it does not require mock objects, although early mockist advice was to mock all dependencies.

## Unreadable Tests

Pseudo English like syntax (DSL) superficially reads well, but inhibits understanding. What things are important? What's actually being tested? You have to work backwards up the test to see what is being checked.



OutputInput,  
Process

Tests with mock objects deviate from the (arguably) more natural 'assemble, act, assert', 'given, when, then' format that is common to most tests. I expect to be able to read sequential code top to bottom.

# Implementation Coupling

Test code restates implementation code rather than checking externally visible state. It assumes I care about the interactions, mostly I don't and shouldn't care. What I care about the end result.

Over-specified by  
default

Mock frameworks "over specify" by default, i.e. a test fails if a method on a mock object is invoked that wasn't specified in the test.

# Brittle

By stating the implementation expectations you end up with brittle tests where a change in the implementation breaks tests. Middle-man objects are particularly problematic, e.g. Rails controllers. `assert assigns[:object]`.

## Inhibit Refactoring

As a result of checking interaction rather than state, they can fail on behaviour (i.e. state) preserving changes. Requires test and implementation to be updated. Tests are the safety net for refactoring, false negatives are bad.

## Lower Confidence

I've seen situations with heavily mocked systems where integration tests or the application break despite all unit tests passing. Requires very high discipline, e.g. J B Rainsberger style to mitigate.



Don't mock that

Mockists are gradually restricting their advice on where to use mocks. Used to be mock all collaborators, e.g. Mock Visual Age. Now, don't mock types you don't own, don't mock value objects, don't over specify, stub commands expect queries.

# Stubs

I predict that expectation setting mock frameworks will gradually be replaced by stubs and test spies, which more cleanly separates input and asserts.

# Spys

A test spy is similar to a mock but records interactions for later checking. Much more like what you write when you hand code stubs.



It's already  
happening

A lot of people have had similar experiences with mocks. In Java land Mockito (despite the name) is a test spy framework that's gaining popularity. In Ruby I'm starting to see the same complaints about mocks, and test spy frameworks are starting to emerge, e.g. `not_a_mock` an rspec extension.



More readable

Follows the more natural, 'assemble, act, assert' style. Easy to see what's input and what's being checked

Under-specified by  
default

It's up to the person writing the test to decide what to check, if you don't care about it it won't cause the test to fail.



**CODE**Dexterity