

Mocks SUCK

and what to do about it



CODEDexterity

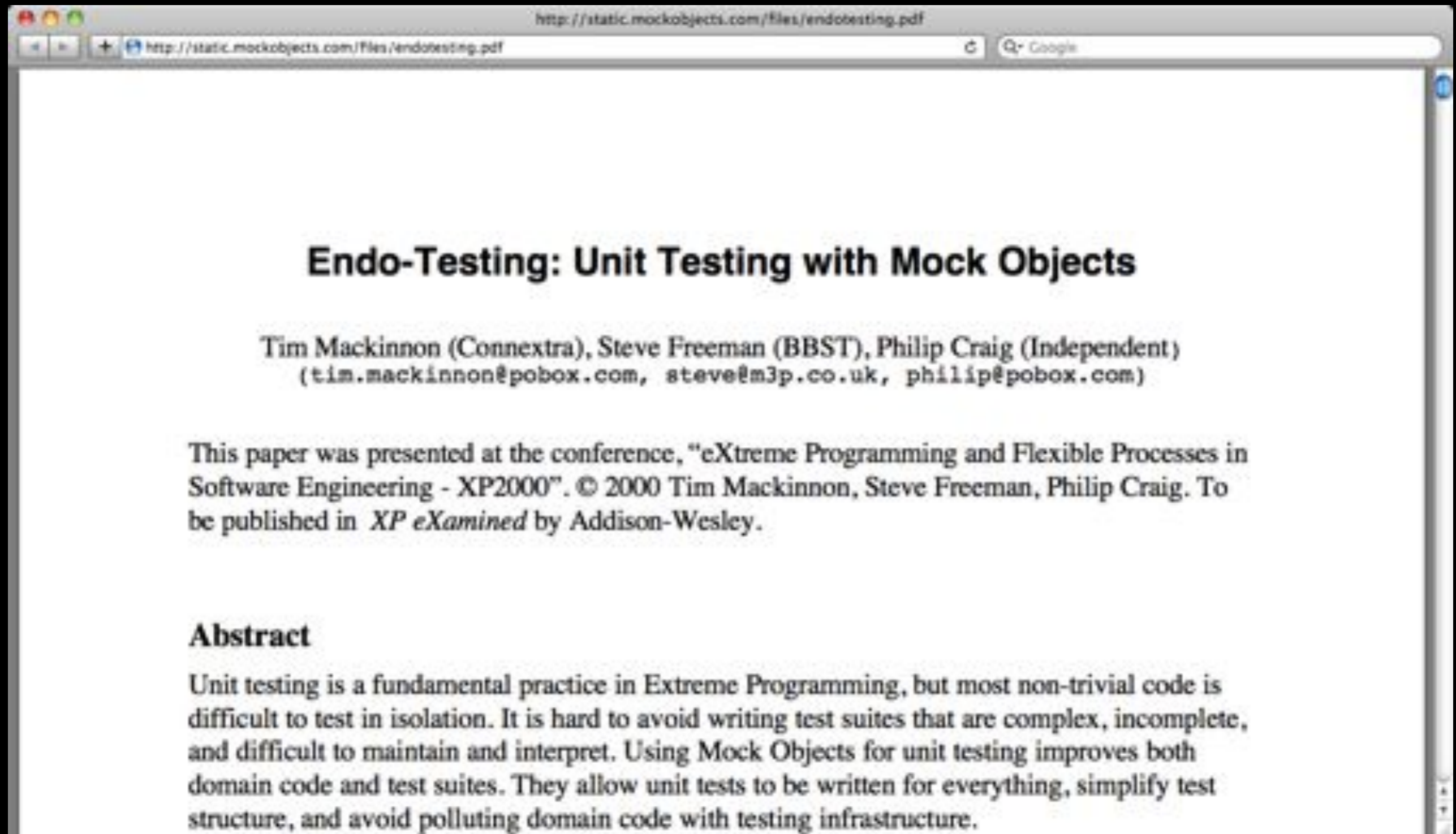
Mocks suck



Y2K



Endotesting



The image shows a screenshot of a web browser window. The address bar contains the URL <http://static.mockobjects.com/files/endotesting.pdf>. The page content is as follows:

Endo-Testing: Unit Testing with Mock Objects

Tim Mackinnon (Connextra), Steve Freeman (BBST), Philip Craig (Independent)
(tim.mackinnon@pobox.com, steve@m3p.co.uk, philip@pobox.com)

This paper was presented at the conference, "eXtreme Programming and Flexible Processes in Software Engineering - XP2000". © 2000 Tim Mackinnon, Steve Freeman, Philip Craig. To be published in *XP eXamined* by Addison-Wesley.

Abstract

Unit testing is a fundamental practice in Extreme Programming, but most non-trivial code is difficult to test in isolation. It is hard to avoid writing test suites that are complex, incomplete, and difficult to maintain and interpret. Using Mock Objects for unit testing improves both domain code and test suites. They allow unit tests to be written for everything, simplify test structure, and avoid polluting domain code with testing infrastructure.

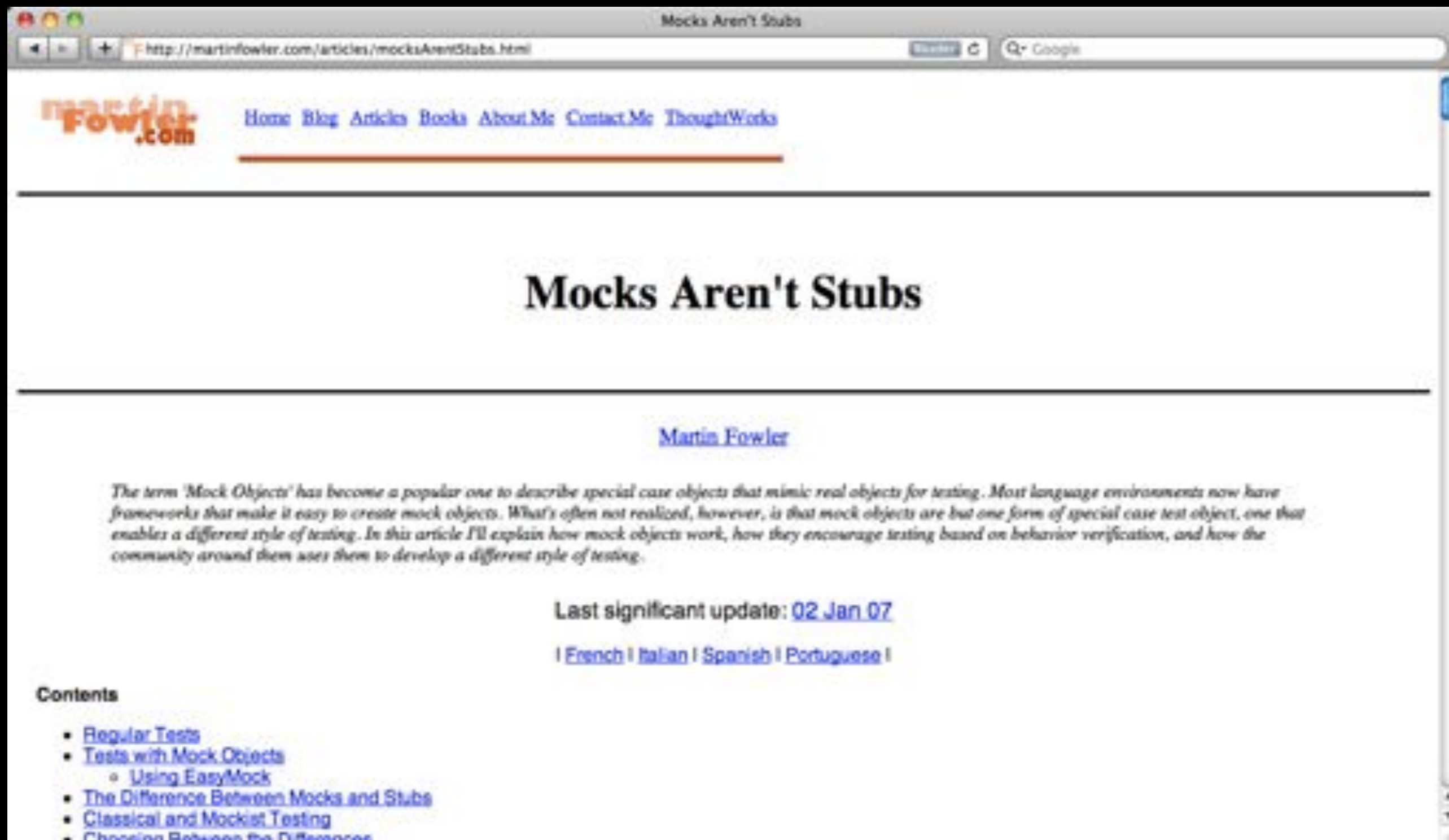
Catchy



mock objects

MockObjects™

Mocks aren't stubs



The screenshot shows a web browser window with the title "Mocks Aren't Stubs". The address bar contains the URL "http://martinfowler.com/articles/mocksArenStubs.html". The browser's search bar shows "Google". The page header includes the "martinfowler.com" logo and navigation links: "Home", "Blog", "Articles", "Books", "About Me", "Contact Me", and "ThoughtWorks". A horizontal line separates the header from the main content area. The main heading is "Mocks Aren't Stubs". Below the heading is the author's name, "Martin Fowler". A paragraph of text follows, starting with "The term 'Mock Objects' has become a popular one to describe special case objects that mimic real objects for testing. Most language environments now have frameworks that make it easy to create mock objects. What's often not realized, however, is that mock objects are but one form of special case test object, one that enables a different style of testing. In this article I'll explain how mock objects work, how they encourage testing based on behavior verification, and how the community around them uses them to develop a different style of testing." Below this paragraph is the text "Last significant update: 02 Jan 07" and a list of language links: "French", "Italian", "Spanish", and "Portuguese". At the bottom left, there is a "Contents" section with a list of links: "Regular Tests", "Tests with Mock Objects" (with a sub-link "Using EasyMock"), "The Difference Between Mocks and Stubs", "Classical and Mockist Testing", and "Choosing Between the Differences".

martinfowler.com

[Home](#) [Blog](#) [Articles](#) [Books](#) [About Me](#) [Contact Me](#) [ThoughtWorks](#)

Mocks Aren't Stubs

[Martin Fowler](#)

The term 'Mock Objects' has become a popular one to describe special case objects that mimic real objects for testing. Most language environments now have frameworks that make it easy to create mock objects. What's often not realized, however, is that mock objects are but one form of special case test object, one that enables a different style of testing. In this article I'll explain how mock objects work, how they encourage testing based on behavior verification, and how the community around them uses them to develop a different style of testing.

Last significant update: [02 Jan 07](#)

[French](#) | [Italian](#) | [Spanish](#) | [Portuguese](#) |

Contents

- [Regular Tests](#)
- [Tests with Mock Objects](#)
 - [Using EasyMock](#)
- [The Difference Between Mocks and Stubs](#)
- [Classical and Mockist Testing](#)
- [Choosing Between the Differences](#)

```
def double(*args)
  declare_double('Double', *args)
end
```

```
# Just like double, but use double
def mock(*args)
  declare_double('Mock', *args)
end
```

```
# Just like double, but use double
def stub(*args)
  declare_double('Stub', *args)
end
```



On the one hand there is a difference in how test results are verified: a distinction between state verification and behavior verification. On the other hand is a whole different philosophy to the way testing and design play together, which I term here as the classical and mockist styles of Test Driven Development.

Martin Fowler, <http://martinfowler.com/articles/mocksArentStubs.html>



On the one hand there is a difference in how test results are verified: a distinction between state verification and **behavior verification**. On the other hand is a whole different philosophy to the way testing and design play together, which I term here as the classical and **mockist styles of Test Driven Development**.

Martin Fowler, <http://martinfowler.com/articles/mocksArentStubs.html>

Behaviour Verification Sucks



```
it "updates the requested book" do
  mock_book = mock(:book)
  Book.should_receive(:find).with("37").and_return(mock_book)
  mock_book.should_receive(:update_attributes).with({:these => 'params'})
  put :update, :id => "37", :book => {:these => 'params'}
end
```

Do I care?



Implementation coupling



Inhibits Refactoring



Brittle



Lowers Confidence



The Tools Suck



(Un)readability



```
@Test public void reportsLostWhenAuctionClosesImmediately() {
    context.checking(new Expectations() {{
        atLeast(1).of(sniperListener).sniperStateChanged(new SniperSnapshot(ITEM_ID, 0, 0, LOST));
    }});

    sniper.auctionClosed();
}
```

Output Input, Process



```
def test_queuing
  during {
    @sut.sendMail(:ignored_sender)
  }.behold! {
    @mailPayload.receives(:attachment_path) { "attachment path" }
    @allContacts.receives(:selectedContact) { "contact" }
    listeners_receive_notification_with_info(News::QueueOutgoingAttachment,
                                             :source_path => "attachment path",
                                             :recipient => "contact")
    listeners_receive_notification(News::CheckOutboxQueue)
  }
end
```

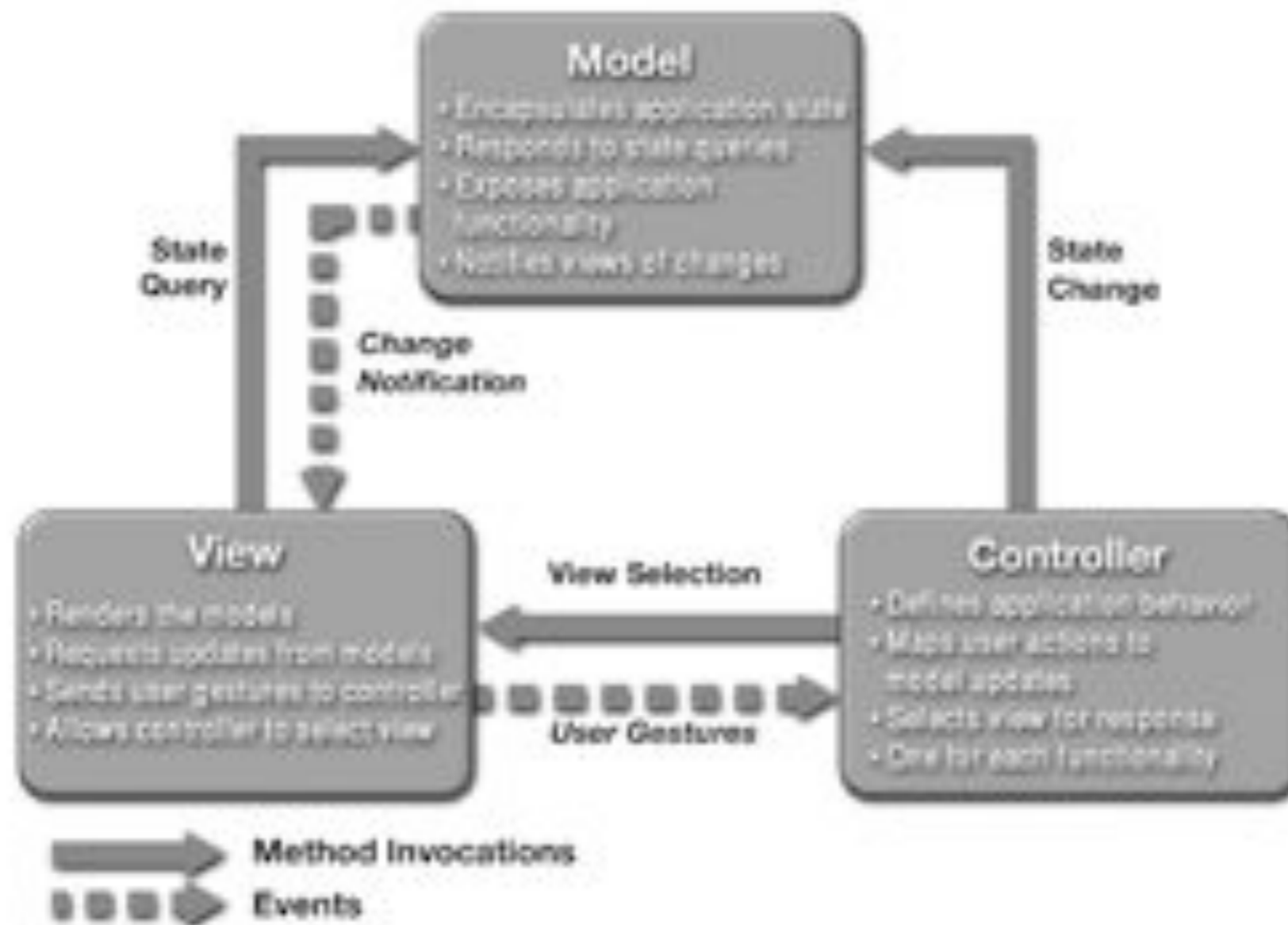
Over-Specified By Default



Mockist TDD Sucks



No Getters and MVC



Thick Glue



Don't Mock that





Don't Mock Types You Can't Change

Steve Freeman, Nat Pryce, Growing Object-Oriented Software, Guided By Tests, p69



Don't Mock Values

Steve Freeman, Nat Pryce, Growing Object-Oriented Software, Guided By Tests, p237



Allow Queries; Expect Commands

Steve Freeman, Nat Pryce, Growing Object-Oriented Software, Guided By Tests, p278



Specify Precisely What Should Happen and No
More

Steve Freeman, Nat Pryce, Growing Object-Oriented Software, Guided By Tests, p274

The Alternative



Use the real thing



```
it "updates the requested book" do
  mock_book = mock(:book)
  Book.should_receive(:find).with("37").and_return(mock_book)
  mock_book.should_receive(:update_attributes).with({:these => 'params'})
  put :update, :id => "37", :book => {:these => 'params'}
end
```

```
it "updates the requested book" do
  a_book = Book.create(:author => 'an author')
  put :update, :id => a_book.to_param, :book => {:author => 'JR Hartley'}
  assert_equal 'JR Hartley', a_book.author
end
```

Stubs



Spies




It's already happening



Not A Mock

😊 [btakita / rr](#)

More readable



Once upon a time

```
// JMock
@Test public void reportsLostWhenAuctionClosesImmediately() {
    context.checking(new Expectations() {{
        atLeast(1).of(sniperListener).sniperStateChanged(new SniperSnapshot(ITEM_ID, 0, 0,
LOST));
    }});

    sniper.auctionClosed();
}
```

```
// Mockita
@Test
public void reportsLostWhenAuctionClosesImmediately() {
    sniper.auctionClosed();
    verify(sniperListener).sniperStateChanged(new SniperSnapshot(ITEM_ID, 0, 0, LOST));
}
```

Under-specified by
default



Time to change



**WRONG WAY
GO BACK**

Thank-you

email: brian@codedexterity.com

twitter: [@bgswan](https://twitter.com/bgswan)

