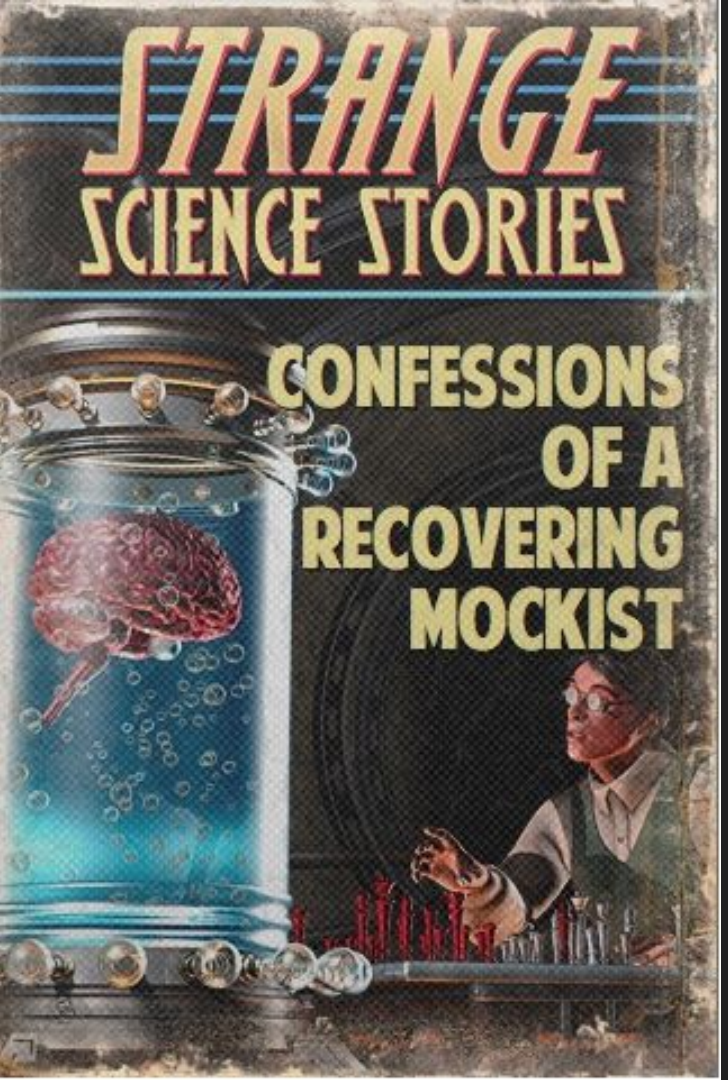


Confessions of a Recovering Mockist

This was me a few years ago

I was pretty Happy

But, At Work...

I mocked everything

I mocked everything

- Domain Classes
- Objects I didn't Own
- Across Service Boundaries
- Utility Classes

And my Software Design Suffered

Modern Tooling:

- made it SO easy to Make Mock Objects
- I didn't Think about the Consequences

Now, I still Mock Some Sociable Classes

But I TRY to Think about it differently Now.

And These are My Confessions

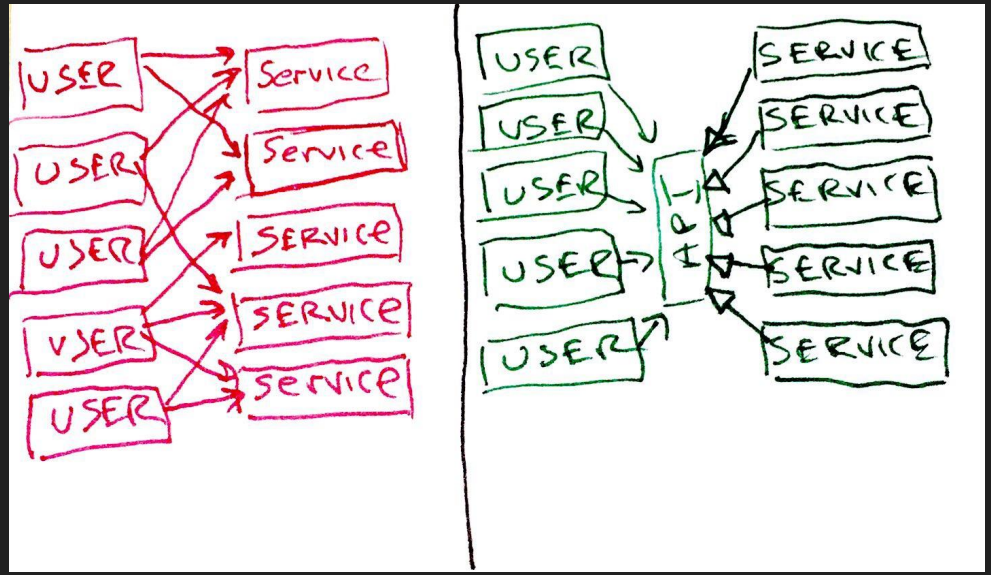
Confession 1:

I used Mocking to Aggressively
Isolated my Unit Tests
 Forcing Coupling between my Tests and
Production Code.

What Does that Mean?

Isolated unit tests Can Couple the tests with the production Code.

I Like to Experiment With "Sociable UNIT Tests"



Confession 2:

I Mocked Not Well Defined Interfaces.

What does That Mean?

Mocking is defining an Explicit Contract
For a collaborating ~~Object~~ Role.

If It is unclear what an Object's Role is it is probably too early to Define a Contract (Mock) for it.

– Paraphrased

Again. What does That Mean?

If it is not Abundantly Clear What Role the Mock is representing you may be:

> coupling to A Specific Implementation
> over-specifying Implementation Details.

I have Seen myself do this, by Mocking:

- Utility Classes, Domain Classes, Translators, Factories, Helpers

Often, I have found that this is coupling My code to A specific Implementation Making it Harder to refactor Concerns

Confession Example

I Like to Think about StubBing and Mocking a Collaborating Object as Something "SACRED"

You ARe defining a Deriving AN CONTRACT/INTERFACE - (almost Implementation) And the ACCEPTANCE CRITERIA
IN the test is that the Mock is Used Correctly

I Like to Think about StubBing and Mocking a Collaborating Object as Something "SACRED"

This is Expensive and Dangerous because Mocks are a Hard to Change and A Regression Danger

I Like to Think about StubBing and Mocking a Collaborating Object as Something "SACRED"

Be Careful

This is why I now advocate:

Mock across architecturally significant boundaries, but not within those boundaries.

This is why I now advocate:

Mock across architecturally significant boundaries, but not within those boundaries.

* Or When You are testing an explicit Contract for an Abstraction.
(Which in Itself Is an Architectural Boundary)

This is why I now advocate:

Mock across architecturally significant boundaries, but not within those boundaries.

* Or When You are testing an explicit Contract for an Abstraction.
(Which in Itself Is an Architectural Boundary)
(You could ARgue)

"Another big benefit of this approach is that it forces you to think through what your significant architectural boundaries are; and enforce them with polymorphic interfaces."

This is why I advocate Only Mocking Explicit Interfaces.

Confession 2:

I Would attempt to use Mocks to Prove correctness

Confession 2:

I Would attempt to use Mocks TDD to Prove correctness

What does that Mean?

"TDD is more about confidence in your code and designs than it is about proving correctness. Proving correctness for everything but mathematical models of algorithms is generally a fool's errand."

What does that Mean?

"TDD is more about confidence in your code and designs than it is about proving correctness. Proving correctness for everything but mathematical models of algorithms is generally a fool's errand."

– David Julia

What does that Mean?

"TDD is more about confidence in your code and designs than it is about proving correctness. Proving correctness for everything but mathematical models of algorithms is generally a fool's errand."

- David Julia Who I am pretty Sure took that From Kent Beck

What does that Mean?

Sometimes, It is Tempting to try AND "prove" that Classes are Implemented Correctly By Testing that They are Collaborating with Other Objects Correctly.

What does that Mean?

Unfortunately, This Commonly leads to Layers of Duplicated Code where:

the Implementation is written:

- Once in the Test
- and Once in the Production Code itself

Confession 1:

I Would Mock Objects I didn't own.

Confession 1:

I Would Mock Objects I didn't own.

* If you Have One take Away:
  Take This

What does That mean?

The Test cannot be used to drive DEsign Of the Object. (The Api Belongs to Someone Else)

What does That mean?

These Mocks are Explicitly Over Specifying (Coupling) you to the One implementation of interaction

What does That mean?

There is No Guarantee that you are using The Library Correctly

What does That mean?

It is Probably a Smell that Your Code is not Decoupled from a Third Party Library

Confession Examples

Confession 5:

I would reimplement Complex Behavior in Mocks.

What does that Mean?

If you find yourself creating a complicated setup for a collaborating object

- Perhaps
  - There is not a clear Boundary?

This May be a Good Candidate For:

Another Kind of Testing Object
- Like a
  - Fake
  - Or a "Dummy"

How I Became a Recovering Mockist

If it seems that The only Way to Test Things is By Mocking Collaborating Objects, I take a Break and Ponder.

How I Became a Recovering Mockist

I try to Make Sure both my PAir and I agree on the VALUE OF A MOCK object.

How I Became a Recovering Mockist

I Consider avoiding pulling in Mocking Libraries
and attempt to Create custom "Fakes" to To Better understand my Abstractions

How I Became a Recovering Mockist

I educated myself on the different types of Testing Doubles and the different philosophies in the TDD community

How I Became a Recovering Mockist

I attend "MockIsts ANONYMOUS" on Tuesdays.

How I Became a Recovering Mockist

I attend "MockIsts ANONYMOUS" on Tuesdays.

* This is a Joke

How I Became a Recovering Mockist

I attend "MockIsts ANONYMOUS" on Tuesdays.

* This is a Joke
  * Unless you Want to start it with me.

https://8thlight.com/blog/uncle-bob/2014/05/10/WhenToMock.html

https://martinfowler.com/articles/mocksArentStubs.html

https://martinfowler.com/bliki/RoleInterface.html

https://github.com/mockito/mockito/wiki/How-to-write-good-tests

https://8thlight.com/blog/eric-smith/2011/10/27/thats-not-yours.html

https://davesquared.net/2011/04/dont-mock-types-you-dont-own.html

http://www.jmock.org/oopsla2004.pdf

https://content.pivotal.io/blog/tdd-midlife-crisis-white-box-testing-refactoring-tests