NMock2
A quick introduction

"If we want everything to remain as it is, it will be necessary for everything to change."
Giuseppe Tomasi di Lampedusa

Presented by
Jason Kerney & Woody Zuill
Software Engineers

wzuill@yahoo.com
http://zuill.us/WoodyZuill

Some Company in Carlsbad, CA

What are Mock Objects for?

- Testing Behavior
- They provide a way to say
  - "I EXPECT" some specific thing to happen
  - when the code I am testing
  - makes a call to a method or property
  - of some other object
- If the "EXPECTATION" is not met during the test, it lets us know that the test failed.

A simple NUnit test using NMock2

```
[Test]
public void testSomethingUseful()
{
     Mockery mockery = new Mockery();
     IDoc docMock = mockery.NewMock< IDoc>();

     Expect.Once.On ( docMock )
          .Method ( “MethodName”)
          .With ( SomeParameterValue )
          .Will ( Return.Value ( SomeReturnValue ) );

    // do something with your actual code under test here using
    // using docMock in place of the DOC

    mockery.VerifyAllExpectationsHaveBeenMet();
}
```

(DOC = “depended-on-component”)

What is the problem!?

- Things are good – we (hopefully) have a lot of NUnit tests in place and that has been really helpful, but…
- Sometimes a bug is discovered that we can’t test for just by checking the state of the OUT (Object-Under-Test)
  - We need to do more than just verify STATE – we need to test BEHAVIOR.
  - Assertions don’t tell us if calls made to DOCs (Depended-On-Components) were done correctly – or even made at all.

Other problems…

- The OUT needs input from DOC
  - We don't have a way to control that "indirect input".
- DOC has lots of dependencies of its own
- What this means is, we need Isolation
  - We want it, but can't get it without controlling the DOCs or replacing them with test doubles.

(OUT = "object under test"

A few other considerations …

- The context in which code works determines its behavior.
  - We must control this context in order to test the code.
  - That means controlling all DOCs of the code uses
- The behaviors we need to test might be unobservable from the outside.
  - To observe this behavior, we must peer "inside" or "behind" the OUT

O

We need a mechanism to do this

- One approach is to use a Mock Object to replace a depended-on-component
- Mock Objects are a type of Test Double
- A “Test Double” is any kind of testing object used in place of a real object
- The Benefits of Doubles:
  - We get Isolation
    - By replacing dependencies with something we control
  - Speed
    - The tests will run as fast as possible
  - Dependable results
    - We control the inputs

The Four Test Doubles

- Meszaros identifies 4 types of Test Doubles:
  - Dummy: passed around but never actually used
  - Stub: provides canned answers to calls
  - Fake: a working implementation that does something, and in some cases can be inspected after use to verify state.
  - Mocks: test behavior through the use of programmed expectations to set up and verify the calls expected to be received

From Meszaros book –
xUnit Test Patterns: Refactoring Test Code

Endo-Testing

- Endo-Testing: Unit Testing with Mock Objects
  - By Tim MacKinnon, Steve Freeman, Philip Craig
  - Presented at the XP2000
  - Published in XP eXamined by Addison-Wesley
  - This is the idea of testing from the inside
  - The same guys who wrote jMock

Ways to Mock

- At least three choices:
  - Hand coded static mocks
  - Static mocks generated with a tool
  - Dynamic Mock object library like NMock, which is what we are covering here today

O

How Does a Dynamic Mock Object Work?

- A dynamic mock object takes on the interface of another object
- We code *expectations* which specify how we EXPECT our OUT to interact with the mock object
- The mock object is then substituted for the real object
- It notifies us if any of the expectations are violated.
- Also, a mock object can act as a *stub*.

O

When to consider "Mocking"

- When behavior cannot be verified with Assertions alone
- When we need to verify how the OUT uses some DOC
- When observing state does not prove a DOC was used
- When methods for querying state don't exist
- When we need to control input from a DOC
- When the DOC is difficult to set up
- When the DOC has behavior that is hard to cause
- When the DOC is slow
- When the DOC does not yet exist (hasn't been coded yet)
- For exploratory work in creating characterization tests.

- CAUTION: Just because you can doesn't mean you should. Learn to use mocks - it's good to have them in your toolbox - but don't over do it.

O

What Is NMock2? (at NMock.org)

- A Dynamic Mock Object Library
  - A port (more or less) of jMock
- Free (Open source)
- Uses a “conversational” style to define expectations
- Uses a Fail Fast approach - allowing you to easily pinpoint the exact point the test failed
- Error messages clearly show the reason for the failure (in most cases…).

O

Using NMock2

- Get NMock2.dll (NMock.org)
- Put the dll in your system
- Reference it from your project
- Include a using directive for NMock2 in your NUnit test class

O

Questions?

- Plese speak up, I can't hear you.

wzuill@yahoo.com