

“Newbie Nugget” Unit Testing with Mock

Daryl Spitzer
daryl.spitzer@gmail.com

Daryl Spitzer

- Software Engineer, VMware (since August 2008)
- started playing with Python less than 5 years ago
- working with it full-time for a little more than 2 years

I *won't* cover

- Why use mocks?
- Types of “Test Doubles”
 - (Dummy, Fakes, Stubs, Mocks)
- Test-Driven Development
 - “Classical” and “Mockist” Testing
- Comparison to other mocking libraries

I will cover

- What is Mock?
- A (very) quick overview of Mock's features
- Two examples (time permitting)

Mock

- A mocking library for writing unit tests
- Latest version: 0.3.1 (December 2007)
- action -> assertion
 - (not record -> replay)

Mock's features

- Mock objects can be used for:
 - Patching methods
 - Recording method calls on objects

```
def test_method_calls_something(self):  
    real = ProductionClass()  
    real.something = Mock()  
  
    real.method()  
  
    self.assertTrue(real.something.called,  
                    "method didn't call something")
```

Mock's features

- call information is available
- `call_args_list` tests how many times mock called *and* arguments for each call

```
self.assertEqual(real.something.call_count, 1,  
                 "something called incorrect number of times")
```

```
self.assertEqual(real.something.call_args, (args, kwargs),  
                 "something called with incorrect arguments")
```

```
self.assertEqual(real.something.call_args_list,  
                 [(args1, kwargs1), (args2, kwargs2)],  
                 "something called with incorrect arguments")
```

Mock's features

- calling methods on a Mock object creates them
- an `AttributeError` is raised if any other methods than those specified are called

```
def test_closer(self):  
    real = ProductionClass()  
    mock = Mock(methods=[ 'close' ])  
  
    real.closer(mock)  
  
    self.assertTrue(mock.close.called,  
                    "closer didn't call close")
```

Mock's features

- the `method_calls` attribute can also be used to verify the called methods

```
>>> mock = Mock()
>>> mock.method1()
>>> mock.method2('foo', x=42)
>>> mock.method_calls
[('method1', (), {}), ('method2', ('foo',), {'x': 42})]
```

Mock's features

- can create a Mock from an existing object (so tests can be coupled to real code instead of mocks)

```
>>> mock = Mock(spec=SomeClass)
```

```
>>>
```

```
>>> mock.old_method()
```

```
Traceback (most recent call last):
```

```
...
```

```
AttributeError: object has no attribute 'old_method'
```

Mock's features

- return values

```
>>> mock = Mock()
>>> mock.return_value = 3
>>> mock()
3
>>> mock.method.return_value = 4
>>> mock.method()
4
>>> yamock = Mock()
>>> mock.method2.return_value = yamock
>>> mock.method2().foo('bar')
>>> mock.method_calls
[('method', (), {}), ('method2', (), {})]
>>> yamock.method_calls
[('foo', ('bar',), {})]
```


Mock's features

- patch decorator makes it easy to patch a class or module attribute

```
mock1 = Mock()  
mock2 = Mock()
```

```
@apply
```

```
@patch(SomeClass, 'class_method', mock1)
```

```
@patch(SomeClass, 'static_method', mock2)
```

```
def test():
```

```
    SomeClass.class_method()
```

```
    SomeClass.static_method()
```

```
self.assertTrue(mock1.called, "class_method not called")
```

```
self.assertTrue(mock2.called, "static_method not called")
```

Mock's features

- can patch modules by supplying a name (including `'__builtin__'`)

```
mock = Mock()
mock.return_value = sentinel.Handle

@patch('__builtin__', 'open', mock)
def test():
    return open('filename', 'r')

handle = test()

self.assertEqual(mock.call_args, (('filename', 'r'), {}),
                  "open not called correctly")
self.assertEqual(handle, sentinel.Handle,
                  "incorrect file handle returned")
```

Mock's features

- omit the third argument to patch to patch with a mock (which is passed in as an extra argument to the decorated function)

```
@patch('Module', 'ClassName1')
@patch('Module', 'ClassName2')
def testMethod(self, mockClass1, mockClass2):
    ClassName1()
    ClassName2()
    self.assertEqual(mockClass1.called,
                     "ClassName1 not patched")
    self.assertEqual(mockClass2.called,
                     "ClassName2 not patched")
```


Example 2

```
@patch('__builtin__', 'open')
def test_render(self, open_stub):
    TEMPLATE_FILE_CONTENTS = r"""producer: %(producer)s
producer_branch: %(producer_branch)s
new_CLN: %(new_CLN)s
consumer_branch: %(consumer_branch)s
contact: %(contact)s
"""

    file_stub = Mock()
    file_stub.read.return_value = TEMPLATE_FILE_CONTENTS
    open_stub.return_value = file_stub

    template = ComponentTemplate(consumer_dir='/ignored',
                                component_name='ignored',
                                context_dict=self.CONTEXT_DICT)

    result = template.render()

    self.assertEqual(result,
                      ''.join((template.HEADER, TEMPLATE_FILE_CONTENTS))
                      % self.CONTEXT_DICT)
```

References

- <http://www.voidspace.org.uk/python/mock.html>
- <http://www.voidspace.org.uk/python/articles/mocking.shtml>
- http://en.wikipedia.org/wiki/Mock_Object
- <http://martinfowler.com/articles/mocksArentStubs.html>