Lab 2

Software Testing 2023

2023/03/02

#whoami

- Software Quality Lab @ EC547
- TA
  a. 蔡惠喬
     - hctsai.cs10@nycu.edu.tw
  b. 陳舜寧
     - xdev11.cs11@nycu.edu.tw

GitHub Repo

- <student_id>-ST-2023
- Add collaborators
  - XDEv11, chameleon10712, skhuang

Mock

Mocks

- Stubs
  - Provide a canned response to method calls
- Spy
  - Real objects that behave like normal except when a specific condition is met
- Mocks
  - Verifies behavior (calls) to a method

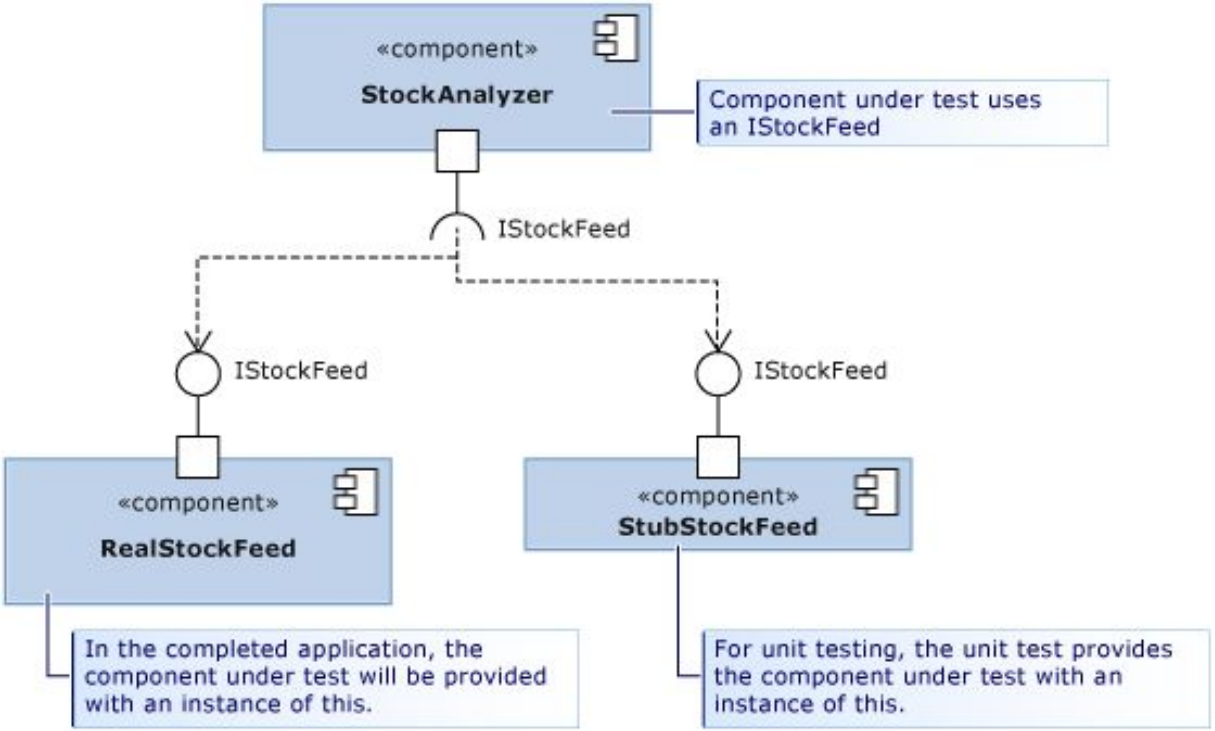
Mocks

- Eliminates dependencies in the class under test

```
#!/usr/bin/env python

def foo(x):
    y = bar(x)
    if y > 10:
        return x+y
    retun x-y
```

Stub


«component»
StockAnalyzer
Component under test uses an IStockFeed
IStockFeed
IStockFeed
IStockFeed
«component»
RealStockFeed
«component»
StubStockFeed
In the completed application, the component under test will be provided with an instance of this.
For unit testing, the unit test provides the component under test with an instance of this.

unittest.mock

The Mock Object

Python >>>

```
>>> from unittest.mock import Mock
>>> mock = Mock()
>>> mock
<Mock id='4561344720'>
```

The Mock Object

Python

```
# Pass mock as an argument to do_something()
do_something(mock)

# Patch the json library
json = mock
```

Lazy Attributes and Methods

A `Mock` must simulate any object that it replaces. To achieve such flexibility, it creates its attributes when you access them:

Python >>>

```
>>> mock.some_attribute
<Mock name='mock.some_attribute' id='4394778696'>
>>> mock.do_something()
<Mock name='mock.do_something()' id='4394778920'>
```

Lazy Attributes and Methods

Python >>>

```
>>> json = Mock()
>>> json.dumps()
<Mock name='mock.dumps()' id='4392249776'>
```

Lazy Attributes and Methods

Python >>>

```
>>> json = Mock()
>>> json.loads('{"k": "v"}').get('k')
<Mock name='mock.loads().get()' id='4379599424'>
```

return_value

Set this to configure the value returned by calling the mock:

```
>>> mock = Mock()
>>> mock.return_value = 'fish'
>>> mock()
'fish'
```

```python
import datetime
from unittest.mock import Mock

# Save a couple of test days
tuesday = datetime.datetime(year=2019, month=1, day=1)
saturday = datetime.datetime(year=2019, month=1, day=5)

# Mock datetime to control today's date
datetime = Mock()

def is_weekday():
    today = datetime.datetime.today()
    # Python's datetime library treats Monday as 0 and Sunday as 6
    return (0 <= today.weekday() < 5)

# Mock .today() to return Tuesday
datetime.datetime.today.return_value = tuesday
# Test Tuesday is a weekday
assert is_weekday()
# Mock .today() to return Saturday
datetime.datetime.today.return_value = saturday
# Test Saturday is not a weekday
assert not is_weekday()
```

side_effect()

```
>>> mock = Mock(side_effect=KeyError('foo'))
>>> mock()
Traceback (most recent call last):
  ...
KeyError: 'foo'
```

side_effect()

```
>>> values = {'a': 1, 'b': 2, 'c': 3}
>>> def side_effect(arg):
...     return values[arg]
...
>>> mock.side_effect = side_effect
>>> mock('a'), mock('b'), mock('c')
(1, 2, 3)
>>> mock.side_effect = [5, 4, 3, 2, 1]
>>> mock(), mock(), mock()
(5, 4, 3)
```

patch

```
>>> @patch.object(SomeClass, 'class_method')
... def test(mock_method):
...     SomeClass.class_method(3)
...     mock_method.assert_called_with(3)
...
>>> test()
```

patch

```
>>> with patch.object(ProductionClass, 'method', return_value=None) as mock_method:
...     thing = ProductionClass()
...     thing.method(1, 2, 3)
...
>>> mock_method.assert_called_once_with(1, 2, 3)
```

patch

```
class unittest.mock.Mock(spec=None, side_effect=None, return_value=DEFAULT,
wraps=None, name=None, spec_set=None, unsafe=False, **kwargs)
```

- *wraps*: Item for the mock object to wrap. If *wraps* is not `None` then calling the Mock will pass the call through to the wrapped object (returning the real result). Attribute access on the mock will return a Mock object that wraps the corresponding attribute of the wrapped object (so attempting to access an attribute that doesn't exist will raise an `AttributeError`).

Nesting Patch Decorators

```
>>> from unittest.mock import patch
>>> @patch('module.ClassName2')
... @patch('module.ClassName1')
... def test(MockClass1, MockClass2):
...     module.ClassName1()
...     module.ClassName2()
...     assert MockClass1 is module.ClassName1
...     assert MockClass2 is module.ClassName2
...     assert MockClass1.called
...     assert MockClass2.called
...
>>> test()
```

call_count

An integer telling you how many times the mock object has been called:

```
>>> mock = Mock(return_value=None)
>>> mock.call_count
0
>>> mock()
>>> mock()
>>> mock.call_count
2
```

call_args_list

This is a list of all the calls made to the mock object in sequence (so the length of the list is the number of times it has been called). Before any calls have been made it is an empty list. The `call` object can be used for conveniently constructing lists of calls to compare with `call_args_list`.

```
>>> mock = Mock(return_value=None)
>>> mock()
>>> mock(3, 4)
>>> mock(key='fish', next='w00t!')
>>> mock.call_args_list
[call(), call(3, 4), call(key='fish', next='w00t!')]
>>> expected = [(), ((3, 4),), ({'key': 'fish', 'next': 'w00t!'},)]
>>> mock.call_args_list == expected
True
```

Lab 2

The Hunger Games

The nation Utopian select several children each year to fight to death in the annual Hunger Games. Contrast to the old games, the government wants to utilize new technology to assist management. Therefore, the ruler recently asked to develop a random selector app for this significant activity. As a test engineer, the government command you to test for this app.

The Hunger Games

The procedure of the app:

```
First, the app will randomly select a child.
If the child is selected, the app will keep selecting until it finds a new kid.
For those who are selected, the app will write and send mails with the text "Congrats, <name>!".
```

Since the app is unfinished and may have several bugs, you need to:

```
(1) Stub the name list of the children.
(2) Mock the random procedure, and test its correctness.
(3) Spy on the MailSystem, and check if the mail is correctly written and sent.
    The total amount of mails shall match the number of selected children.
```

Requirements

- In `setUp()`, stub the name list of the children.
  - Set name list as "William, Oliver, Henry, Liam"
  - Set the selected ones as "William, Oliver, Henry"
- To test class `Application`, you need to:
  - Mock `get_random_person()`, return values as follows: "William, Oliver, Henry, Liam".
  - Assure not to select the ones who are already selected.
  - Examine the result of `select_next_person()` using `assertEqual`.
- To test class `MailSystem`, you need to:
  - Finish `fake_mail()` and print the mail context.
  - Spy on `send()` and `write()`.
  - Examine the call count of `send()` and `write()` using `assertEqual`.
- Notes
  - The real mail system is well structured and complicated, `app.py` only write pseudo code to represent it. For MailSystem, you only need to test the logic between the call dependency.

```
python3 -m unittest app_test.py 2>/dev/null
python3 -m unittest app_test.py -b -v
```

or

```
python3 -m unittest app_test.py
```

Test with unittest

```
▶ Run cd Lab02
--select next person--
Liam selected
--notify selected--
Congrats, William!
Congrats, Oliver!
Congrats, Henry!
Congrats, Liam!


[call('William'), call('Oliver'), call('Henry'), call('Liam')]
[call('William', 'Congrats, William!'),
 call('Oliver', 'Congrats, Oliver!'),
 call('Henry', 'Congrats, Henry!'),
 call('Liam', 'Congrats, Liam!')]
test_app (app_test.ApplicationTest) ... ok

----------------------------------------------------------------------
Ran 1 test in 0.002s

OK
```

Spec & Code

- NYCU-Software-Testing-2023/Lab02/

Submission

Submission

- Use Lab02-CI.yml, and add Lab02 status badge in your README
- Please submit your Github repo <student_id>-ST-2023 commit URL to E3
- commit URL
    - refer to Lab 1 submission

Reference

- https://docs.python.org/3/library/unittest.mock.html#nesting-patch-decorators
- https://learn.microsoft.com/en-us/visualstudio/test/using-stubs-to-isolate-parts-of-your-application-from-each-other-for-unit-testing?view=vs-2022&tabs=csharp
- https://www.baeldung.com/mockito-spy#simple-spy-example
- https://realpython.com/python-mock-library/