# Testing the Untestable

A beginner's guide to mock objects

@andrewburrows

- London based systematic hedge fund since 1987
- $19.2bn Funds Under Management (2016-03-31)
- We are active in 400+ markets in 40+ countries
- We take ~2bn market data points each day
- https://github.com/manahl/arctic
- 125 people, 22 first languages. And Python!

@manahltech

https://github.com/manahltech

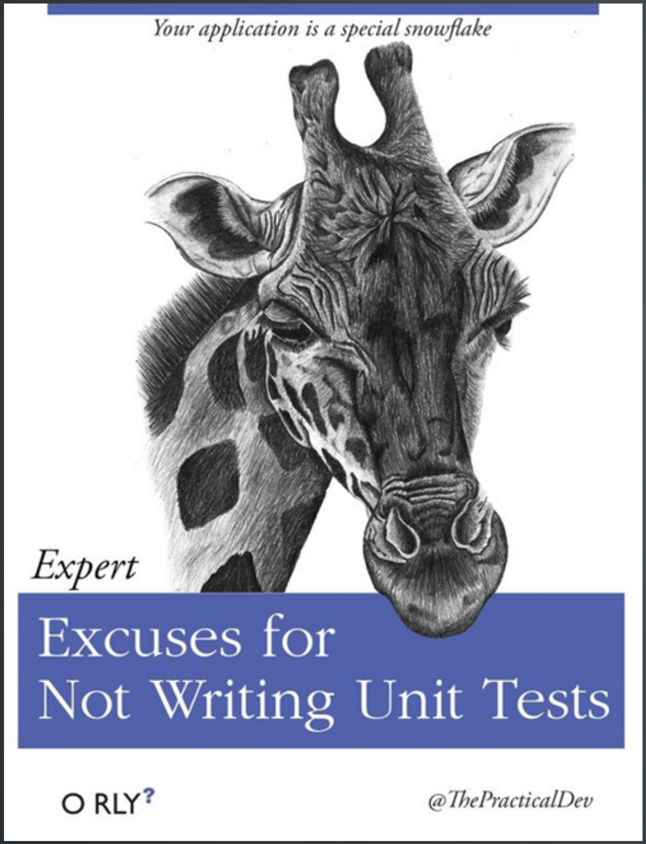
# Testing the Untestable

## A beginner's guide to mock objects

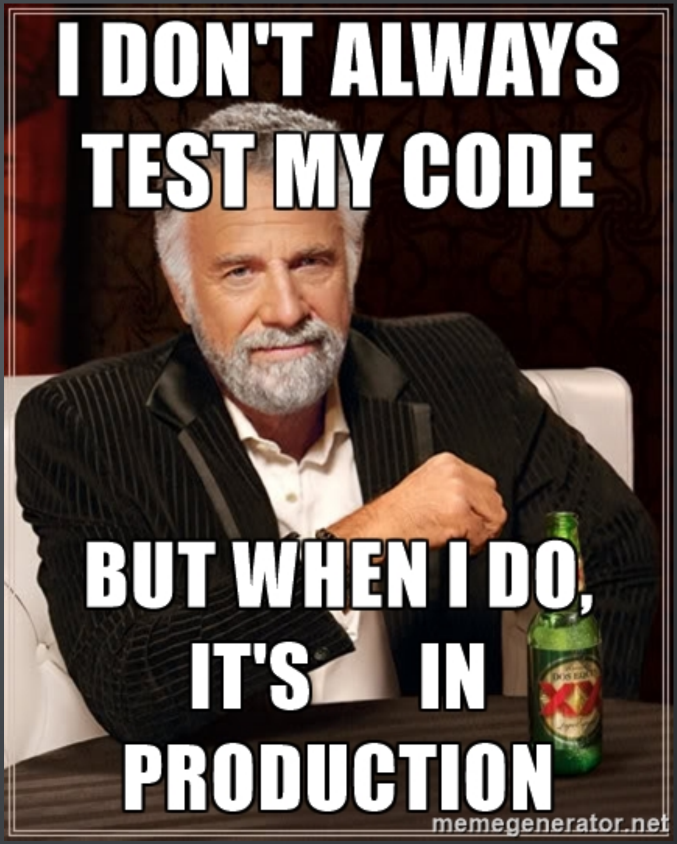
- Example based
- Some theory/definitions
- pytest
- python3

https://github.com/burrowsa/mocking

## Why am I here?


Your application is a special snowflake
Expert
Excuses for Not Writing Unit Tests
O RLY?
@ThePracticalDev

https://twitter.com/thepracticaldev


I DON'T ALWAYS TEST MY CODE
BUT WHEN I DO, IT'S IN PRODUCTION
memegenerator.net

https://memegenerator.net/instance/65198099

## Easy Example

```python
class ConferenceSpeaker(object):
    def __init__(self, name, twitterhandle):
        self.name = name
        self.twitterhandle = twitterhandle

    def greet(self, delegates):
        for delegate in delegates:
            delegate.speakto("Hi my name is {0.name}, follow me"
                             "on twitter @{0.twitterhandle}".format(s
```

# Easy Example

```
class ConferenceSpeaker(object):
    def __init__(self, name, twitterhandle):
        self.name = name
        self.twitterhandle = twitterhandle

    def greet(self, delegates):
        for delegate in delegates:
            delegate.speakto("Hi my name is {0.name}, follow me"
                             "on twitter @{0.twitterhandle}".format(s
```

## System Under Test (SUT)

The "system under test". It is short for "whatever thing we are testing".

# Easy Example

```python
class ConferenceSpeaker(object):
    def __init__(self, name, twitterhandle):
        self.name = name
        self.twitterhandle = twitterhandle

    def greet(self, delegates):
        for delegate in delegates:
            delegate.speakto("Hi my name is {0.name}, follow me"
                             "on twitter @{0.twitterhandle}".format(se
```

# Not so easy

```
import re
import simpletweeter


TWITTER_REGEX = re.compile(".*follow me on twitter @(\w+)")


class ConferenceDelegate(object):
    def __init__(self, credentialsfile):
        self.credentialsfile = credentialsfile

    def speakto(self, message):
        matched = TWITTER_REGEX.match(message)
        if matched:
            simpletweeter.tweet("Amazing talk from @" + matched.group
                                self.credentialsfile)
```

## If it quacks like a duck...

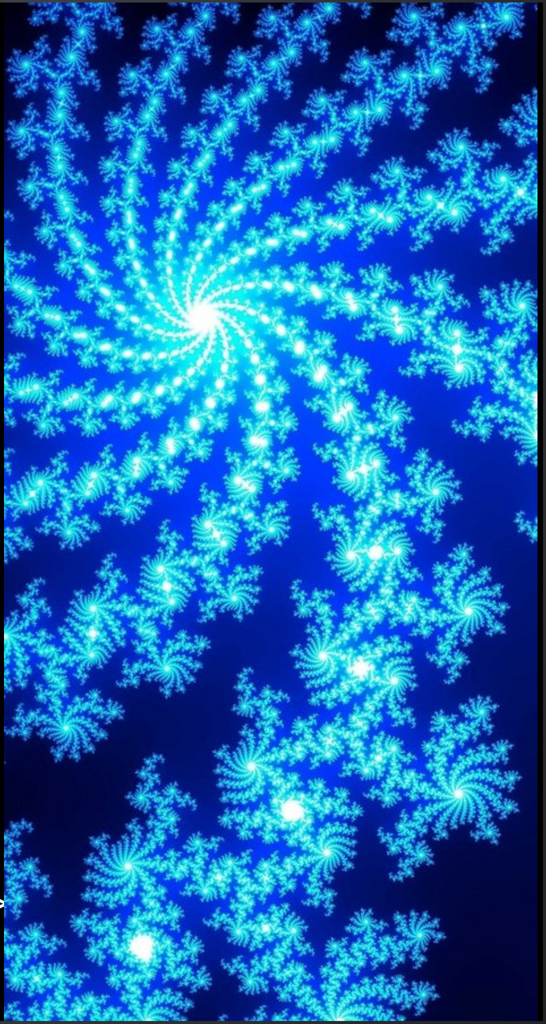
```
from mocking import ConferenceSpeaker


def test_speaker_greets_sole_delegate_no_mocks():
    sut = ConferenceSpeaker("Andy Burrows", "andrewburrows")

    class TestDelegate(object):
        def __init__(self):
            self.calls = []
        def speakto(self, msg):
            self.calls.append(("speakto", msg))

    delegate = TestDelegate()

    sut.greet([delegate])

    assert delegate.calls == [("speakto", "Hi my name is Andy Burro
                               "follow me on twitter @andrewburrows
```

## Mock FTW!!!

```python
from mocking import ConferenceSpeaker, ConferenceDelegate
from unittest.mock import Mock, call


def test_speaker_greets_sole_delegate():
    # Arrange
    sut = ConferenceSpeaker("Andy Burrows", "andrewburrows")
    delegate = Mock()

    # Act
    sut.greet([delegate])

    # Assert
    delegate.speakto.assert_called_once_with("Hi my name is Andy Burrows, "
                                             "follow me on twitter @andrewbu
```

# It's Mocks all the way down

```
>>> from unittest.mock import Mock

>>> my_mock = Mock()
>>> my_mock
<Mock id='56147192'>

>>> my_mock = Mock(name="my_mock")
>>> my_mock
<Mock name='my_mock' id='56191128'>

>>> my_mock.hello
<Mock name='my_mock.hello' id='56146744'>

>>> my_mock()
<Mock name='my_mock()' id='56202912'>

>>> my_mock.a_method(1, 2, 3)
<Mock name='my_mock.a_method()' id='56147192'>
```



## parlez-vous mocks?

Test Double - any pretend object used for testing

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

# parlez-vous mocks?

Test Double - any pretend object used for testing

Fake - e.g. a in memory database used in place of the real DB

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

# parlez-vous mocks?

Test Double - any pretend object used for testing

Fake - e.g. a in memory database used in place of the real DB

Dummy - Dummy value used to pad out an argument list or trace the flow of data through our program. Can not interact with the SUT. **In python we use a sentinel.**

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

# parlez-vous mocks?

Test Double - any pretend object used for testing

Fake - e.g. a in memory database used in place of the real DB

Dummy - Dummy value used to pad out an argument list or trace the flow of data through our program. Can not interact with the SUT. **In python we use a sentinel.**

Mock - Pretend object which records interactions and allows the test code to assert these match expectations.

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

# parlez-vous mocks?

Test Double - any pretend object used for testing

Fake - e.g. a in memory database used in place of the real DB

Dummy - Dummy value used to pad out an argument list or trace the flow of data through our program. Can not interact with the SUT. **In python we use a sentinel.**

Mock - Pretend object which records interactions and allows the test code to assert these match expectations.

Stub - Pretend object which supports limited, canned interactions with the SUT. **In python we use a Mock with a side_effect.**

Spy - see Mock

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

# Assertions

```python
from mocking import ConferenceSpeaker, ConferenceDelegate
from unittest.mock import Mock, call


def test_speaker_greets_sole_delegate():
    # Arrange
    sut = ConferenceSpeaker("Andy Burrows", "andrewburrows")
    delegate = Mock()

    # Act
    sut.greet([delegate])

    # Assert
    delegate.speakto.assert_called_once_with("Hi my name is Andy Burrows, "
                                             "follow me on twitter @andrewbu
```

# Assertions

```
def test_speaker_greets_sole_delegate_v2():
    # Arrange
    sut = ConferenceSpeaker("Andy Burrows", "andrewburrows")
    delegate = Mock()

    # Act
    sut.greet([delegate])

    # Assert
    assert delegate.mock_calls == [call.speakto("Hi my name is Andy Burrows,
                                                "follow me on twitter @andre
```

```
>>> m = Mock()
>>> m.foo('hello')
>>> m.bar('world')
>>> x = m(0)
>>> x.hello(123)
>>> print(m.mock_calls)
[call.foo('hello'), call.bar('world'), call(0), call().hello(123)]
```

# Spec=

```python
def test_speaker_greets_sole_delegate_v3():
    # Arrange
    sut = ConferenceSpeaker("Andy Burrows", "andrewburrows")
    delegate = Mock(spec=ConferenceDelegate)

    # Act
    sut.greet([delegate])

    ...
```

```
>>> m = Mock(spec=ConferenceDelegate)
>>> m.snore(volume="LOUD")
Traceback (most recent call last):
  File "mocking\tests\test_conference_speaker.py", line 83, in <modul
    mock_delegate.snore(volume="LOUD")
  File "unittest\mock.py", line 557, in __getattr__
    raise AttributeError("Mock object has no attribute %r" % name)
AttributeError: Mock object has no attribute 'snore'
```

# Harder Example

```
import re
import simpletweeter


TWITTER_REGEX = re.compile(".*follow me on twitter @(\w+)")


class ConferenceDelegate(object):
    def __init__(self, credentialsfile):
        self.credentialsfile = credentialsfile

    def speakto(self, message):
        matched = TWITTER_REGEX.match(message)
        if matched:
            simpletweeter.tweet("Amazing talk from @" + matched.group
                                self.credentialsfile)
```

# Harder Example

```python
import re
import simpletweeter


TWITTER_REGEX = re.compile(".*follow me on twitter @(\w+)")


class ConferenceDelegate(object):
    def __init__(self, credentialsfile):
        self.credentialsfile = credentialsfile

    def speakto(self, message):
        matched = TWITTER_REGEX.match(message)
        if matched:
            simpletweeter.tweet("Amazing talk from @" + matched.group
                                self.credentialsfile)
```

## Patch

```python
from unittest.mock import sentinel, patch


def test_delegate_tweets_if_message_contains_twitter_handle():
    sut = ConferenceDelegate(sentinel.credentialsfile)

    with patch("simpletweeter.tweet") as mock_tweet:
        sut.speakto("Hi, why not follow me on twitter @manahltech")

    mock_tweet.assert_called_once_with("Amazing talk from @manahltech
                                       sentinel.credentialsfile)
```

```
import re
from simpletweeter import tweet


TWITTER_REGEX = re.compile(".*follow me on twitter @(\w+)")


class ConferenceDelegate(object):
    ...

    def speakto(self, message):
        matched = TWITTER_REGEX.match(message)
        if matched:
            tweet("Amazing talk from @" + matched.groups()[0],
                  self.credentialsfile)
```

```
def test_delegate_tweets_if_message_contains_twitter_handle():
    sut = ConferenceDelegate(sentinel.credentialsfile)

    with patch("mocking.conferencedelegate.tweet") as mock_tweet:
        sut.speakto("Hi, why not follow me on twitter @manahltech")

    mock_tweet.assert_called_once_with("Amazing talk from @manahltech
                                       sentinel.credentialsfile)
```

# Sentinels

```
from unittest.mock import sentinel, patch


def test_delegate_tweets_if_message_contains_twitter_handle():
    sut = ConferenceDelegate(sentinel.credentialsfile)

    with patch("simpletweeter.tweet") as mock_tweet:
        sut.speakto("Hi, why not follow me on twitter @manahltech")

    mock_tweet.assert_called_once_with("Amazing talk from @manahltech
                                       sentinel.credentialsfile)
```

# simpletweeter.py

```python
import tweeterapi
from os.path import expanduser


CREDENTIALS_FILE = expanduser("~/twittercredentials.cfg")


def tweet(msg, credentials_file=CREDENTIALS_FILE):
    """Sends a tweet using the login credentials supplied in a file and
    retries up to 5 times in the even of a failure.

    Args:
        msg (str): The message to be tweeted.
        credentials_file (str): The path of the file containing the login creder
    """
    username, password = _read_credentials(credentials_file)
    t = tweeterapi.Tweeter(username, password)

    for _ in range(5):
        if t.tweet(msg):
            break
    else:
        raise RuntimeError("Unable to tweet")
```

# return_value

```
@patch('tweeterapi.Tweeter')
@patch('simpletweeter._read_credentials', return_value=(sentinel.user
def test_tweet_raises_exception_on_failure(_, mock_tweeter):
    mock_tweeter.return_value.tweet.return_value = False

    with pytest.raises(RuntimeError) as err:
        tweet(sentinel.message, sentinel.credentials_file)

    assert str(err.value) == "Unable to tweet"
```

## @patch

```
@patch('tweeterapi.Tweeter')
@patch('simpletweeter._read_credentials', return_value=(sentinel.user
def test_tweet_raises_exception_on_failure(_, mock_tweeter):
    mock_tweeter.return_value.tweet.return_value = False

    with pytest.raises(RuntimeError) as err:
        tweet(sentinel.message, sentinel.credentials_file)

    assert str(err.value) == "Unable to tweet"
```

4 days ago

## ~~return_values~~ side_effect

```
@patch('tweeterapi.Tweeter')
@patch('simpletweeter._read_credentials', return_value=(sentinel.user
def test_tweet_retries_on_failure(_, mock_tweeter):
    mock_tweeter.return_value.tweet.side_effect = [False, False, True

    tweet(sentinel.message, sentinel.credentials_file)

    mock_tweeter.mock_calls = [call(sentinel.username, sentinel.passw
                               call.tweet(sentinel.message),
                               call.tweet(sentinel.message),
                               call.tweet(sentinel.message)]
```

## side_effect

- sequence
- exception
- function/lambda
  - Effectively makes a stub
  - Occasionally useful
  - "Bad code smell" if all your tests rely heavily on defining side effects

# Making conversation

```python
from simpletweeter import tweet

class ConferenceDelegate(object):
    ...

    def smalltalk(self, delegate):
        if delegate.speakto("Hello.") in ("hello", "hi"):
            if delegate.speakto("Good conference?") in ("yes", "yup", "not bad", "ye
                best_bit = delegate.speakto("What has been your favourite part?")
                delegate.speakto("Really, I didn't go to that.")
                delegate.speakto("Nice chatting with you, gotta go.")

                tweet("Absolutely loved " + best_bit)
```

## Stubs

```
def test_successful_conversation_using_side_effect():
    stranger = Mock(name="stranger")

    def stranger_speakto(msg):
        if msg == "Hello.":
            return "hi"
        elif msg == "Good conference?":
            return "not bad"
        elif msg == "What has been your favourite part?":
            return "a brilliant talk on mocks"
        elif msg == "Really, I didn't go to that.":
            return "shame, it was amazing"
        elif msg == "Nice chatting with you, gotta go.":
            return "laters"

    stranger.speakto.side_effect = stranger_speakto

    delegate = ConferenceDelegate()
    with patch("mocking.conferencedelegate4.tweet") as mock_tweet:
        delegate.smalltalk(stranger)

    mock_tweet.assert_called_once_with("Absolutely loved a brilliant tal
```

# Mockextras

```
def test_successful_conversation():
    s= Mock(name="stranger")
    when(s.speakto).called_with("Hello.").then("hi")
    when(s.speakto).called_with("Good conference?").then("not bad")
    when(s.speakto).called_with("What has been your favourite part?").then("a brilliant t
    when(s.speakto).called_with("Really, I didn't go to that.").then("shame, it was amaz
    when(s.speakto).called_with("Nice chatting with you, gotta go.").then("laters")

    delegate = ConferenceDelegate()
    with patch("mocking.conferencedelegate4.tweet") as mock_tweet:
        delegate.smalltalk(s)

    mock_tweet.assert_called_once_with("Absolutely loved a brilliant talk on mocks")
```

https://github.com/manahl/mockextras/
http://mockextras.readthedocs.org/

## Beware the Dark Side

# Beware the Dark Side

## Over-mocking

### Do Mock

- Webservices
- Sending email
- Database access
- "Production"
- Disc
- Environment vars
- 3rd party APIs
- Randomness
- Time

# Beware the Dark Side

## Over-mocking

### Do Mock

- Webservices
- Sending email
- Database access
- "Production"
- Disc
- Environment vars
- 3rd party APIs
- Randomness
- Time

### Don't mock

- Builtin types
- "Builtin" types
- numpy/pandas
- "Structs"

## Beware the Dark Side

### Over-mocking

?? Everything else ??

### Do Mock

- Webservices
- Sending email
- Database access
- "Production"
- Disc
- Environment vars
- 3rd party APIs
- Randomness
- Time

### Don't mock

- Builtin types
- "Builtin" types
- numpy/pandas
- "Structs"

## Two Schools

- A classical TDD approach is to use real object wherever possible only using a stub/fake/mock when it is difficult to use the real thing.
- The Mockist style prefers to use Mocks for any object with "interesting behaviour".

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

## Over-mocking

Over mocked tests:

- are brittle to changes in the SUT
- are expensive to maintain
- often get thrown away during refactoring
- give mocks a bad name
- are easy to write ;)
- boost coverage stats ;)

# ISO-K1773N5

```
class ConferenceDelegate(object):
    ...

    def rate_talk(self,
                  number_of_kitten_pics,
                  usefulness_of_content,
                  clarity_of_presentation
                  ):
        return number_of_kitten_pics * (usefulness_of_content
                                        + clarity_of_presenta
```

# Overmocked

```
def test_delegate_can_rate_a_talk():
    kittens = MagicMock(name="number_of_kitten_pics")
    usefulness = MagicMock(name="usefulness_of_content")
    clarity = MagicMock(name="clarity_of_presentation")

    delegate = ConferenceDelegate()

    result = delegate.rate_talk(kittens, usefulness, clarity)

    assert result is kittens.__mul__.return_value
    assert kittens.mock_calls == [('__mul__', (usefulness.__add__.return_val
    assert usefulness.mock_calls == [('__add__', (clarity,))]
```

# Phew!

```
@pytest.mark.parametrize("kittens,"
                         "usefulness,"
                         "clarity,"
                         "expected_rating",
                         [(1, 1, 1, 2), # ticks all the boxes
                          (0, 1, 1, 0), # no cats no points
                          (1, 0, 1, 1), # lacking content
                          (1, 1, 0, 1), # lacking clarity
                          (10, 0, 1, 10), # loadz of cats
                          (10, 1, 0, 10), # loadz of cats
                          (1, 10, 1, 11), # great content
                          (1, 1, 10, 11), # great delivery
                          ])
def test_delegate_can_rate_a_talk_no_mocks(kittens, usefulness, clarity, expected_rating
    delegate = ConferenceDelegate()
    assert expected_rating == delegate.rate_talk(kittens, usefulness, clarity)
```

## Summary

- Write tests
- Use mocks they are easy and fun
- patch is a great tool to inject mocks into your code
- I love sentinels - so should you
- Function side_effects are an "occasional treat"
- Never "over mock"

## Questions

https://github.com/burrowsa/mocking

@manahltech

https://github.com/manahltech