Mocks, dummies, stubs & spies:

Successfully isolating the snake

EuroPython 2018
July 26, 2018

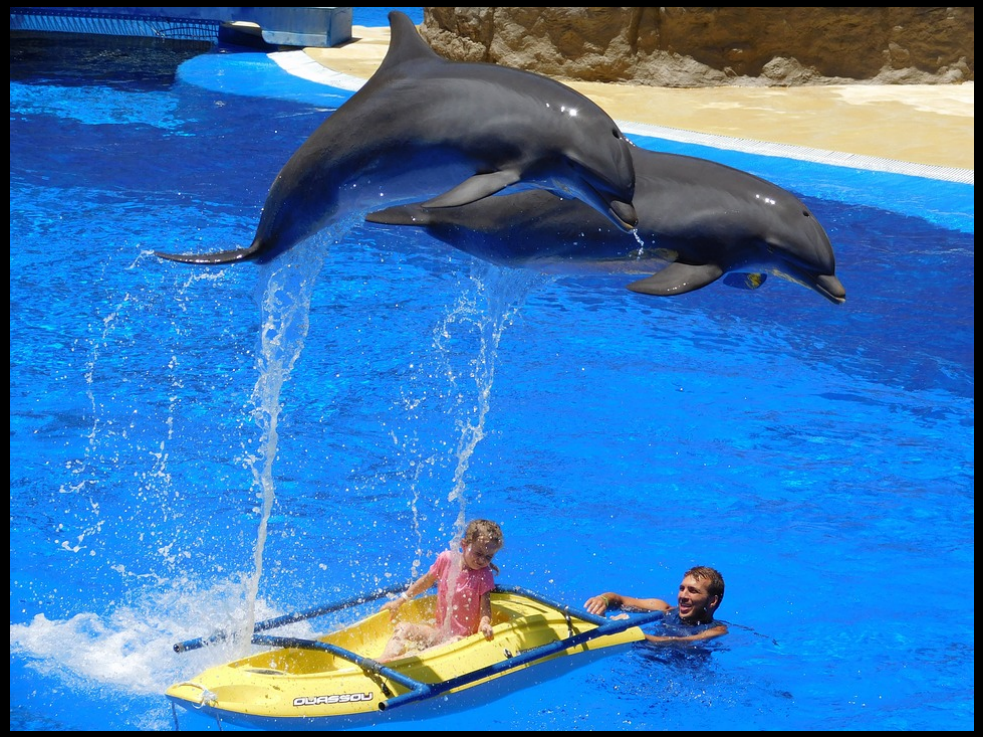
Mario Corchero
Senior Software Developer
@mariocj89

TechAtBloomberg.com

Bloomberg Engineering

What are testing doubles?

An object that *looks like* the real one but the creator is under control of its behavior.



TechAtBloomberg.com

Bloomberg

Engineering

Why we need testing doubles

```
def predict_department_expenses(budget):
    ...
```

How to create testing doubles

```
class MarioDouble():
    @property
    def nationality(self):
        return "Spanish"
    def is_allowed_to_live_in_uk(self):
        return not BREXIT_ENABLED
```

```
unittest.mock.Mock()
```

Filling parameters

A function takes a parameter you just want to ignore.

```
def predict_department_expenses(budget, email_subject):
    ...
```

```
assert EXPECTED_RESULT == predict_department_expenses(INPUT_BUDGET, None)
```

```
assert EXPECTED_RESULT == predict_department_expenses(INPUT_BUDGET, “hihi”)
```

Known as Dummy

Bloomberg

Engineering

Simulating behaviour

```
def predict_department_expenses(budget, email_sender):
    ...
    assert email_sender.is_enabled
    email_sender(...)
    ...
```

```
fake = Mock(return_value="SUCCESS", is_enabled=True)
```

Known as Stub

Bloomberg

Engineering

Simulating behaviour

```
def predict_department_expenses(budget, email_sender):
    ...
    assert email_sender.is_enabled
    email_sender(...)
    ...
```

```
fake = Mock()
```

Known as Stub

Bloomberg

Engineering

Simulating magic methods

```
def predict_department_expenses(budget, email_sender):
    ...
    count_mails_sent = email_sender.send_email(...)
    total_mails_sent += count_mails_sent
    ...
```

```
fake = MagicMock()
```

Known as Stub

Bloomberg

Engineering

More complex behaviour

```
def predict_department_expenses(budget, email_sender):
    ...
    count_mails_sent = email_sender(...)
    if count_mails_sent == 0:
        raise HorribleException(f"{email_sender} did not send emails")
    ...
```

```
stub = Mock(return_value=1)
stub = Mock(return_value=0)
```

```
stub = Mock(side_effect=[2, 10, 20, 0])
```

```
stub = Mock(side_effect=Exception("Failed"))
```

```
stub = Mock(side_effect=print)
```

Known as Stub

Bloomberg

Engineering

Changing internal dependencies

```
import email_service
email_sender = email_service.Sender()

def predict_department_expenses(budget):
    ...
    count_mails_sent = email_sender(...)
    ...
```

```
with unittest.mock.patch('module.to.test.email_sender') as stub:
    stub.return_value = 1
    module.to.test.predict_department_expenses(budget)
```

Known as Stub

Bloomberg

Engineering

Verifying interactions

```
def predict_department_expenses(budget, email_sender):
    ...
    sent_count = email_sender.send_mail(...)
    if not sent_count:
        raise Exception("Failed to send emails")
    ...
```

```
mock = MagicMock()
mock.send_email.return_value = 3
predict_department_expenses(budget)
mock.send_email.assert_called_with(SUBJECT, to=EMAIL)
```

Known as Mock

Using spec

```
mock = Mock(spec=mailing)
mock.send_email() # raises AttributeError
```

Using seal

```
def predict_department_expenses(budget, email_sender):
    ...
    sent_count = email_sender.send_mail(...).response[0].data
    ...
```

```
mock = MagicMock()
email_sender.send_mail(...).response[0].payload = “{}”

predict_department_expenses(budget, mock)

mock.assert_called()
```

Known as Mock

Bloomberg

Engineering

Using seal

```
def predict_department_expenses(budget, email_sender):
    ...
    sent_count = email_sender.send_mail(...).response[0].data
    ...
```

```
mock = MagicMock()
email_sender.send_mail(...).response[0].data = “{}”
seal(email_sender)

predict_department_expenses(budget, mock)

mock.assert_called()
```

Known as Mock

Bloomberg
Engineering

Inspect interaction of a real object

```
def predict_department_expenses(budget, email_sender):
    ...
    email_sender.send_email(...)
    ...
```

```
spy = Mock(wraps=email_sender)
```

```
assert spy.called
args, kwargs = spy.call_args
subject, destination = args
```

Known as Spy

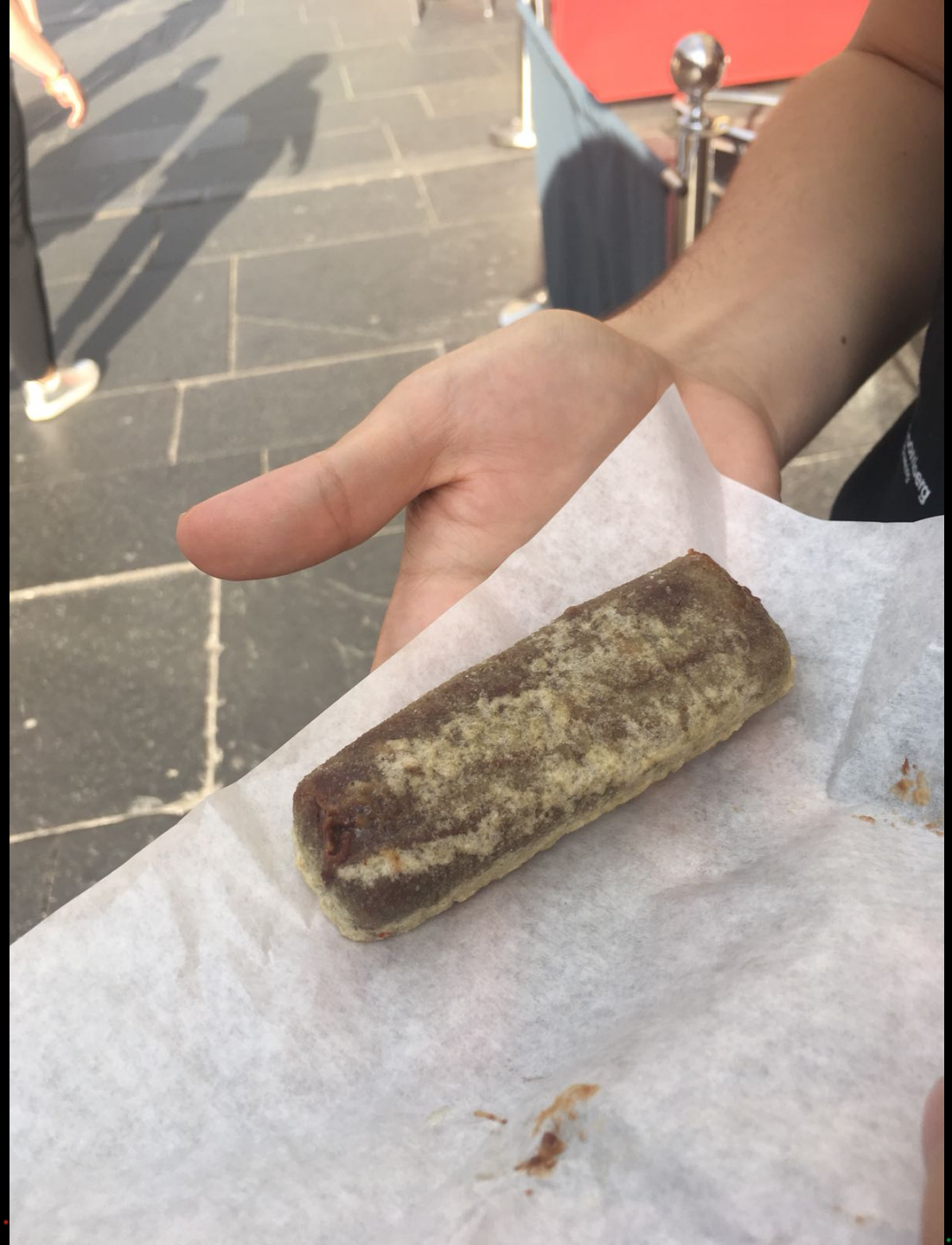
Bloomberg

Engineering

Wrap up & conclusions

- Names: Dummies, Fakes, Stubs, Spies & Mocks
- Unittest.mock helps us create them
- Patch can be used to use testing doubles on internal dependencies
- Use spec or seal to freeze your Mock instances
- Wraps allows you to easily create spies

Take away

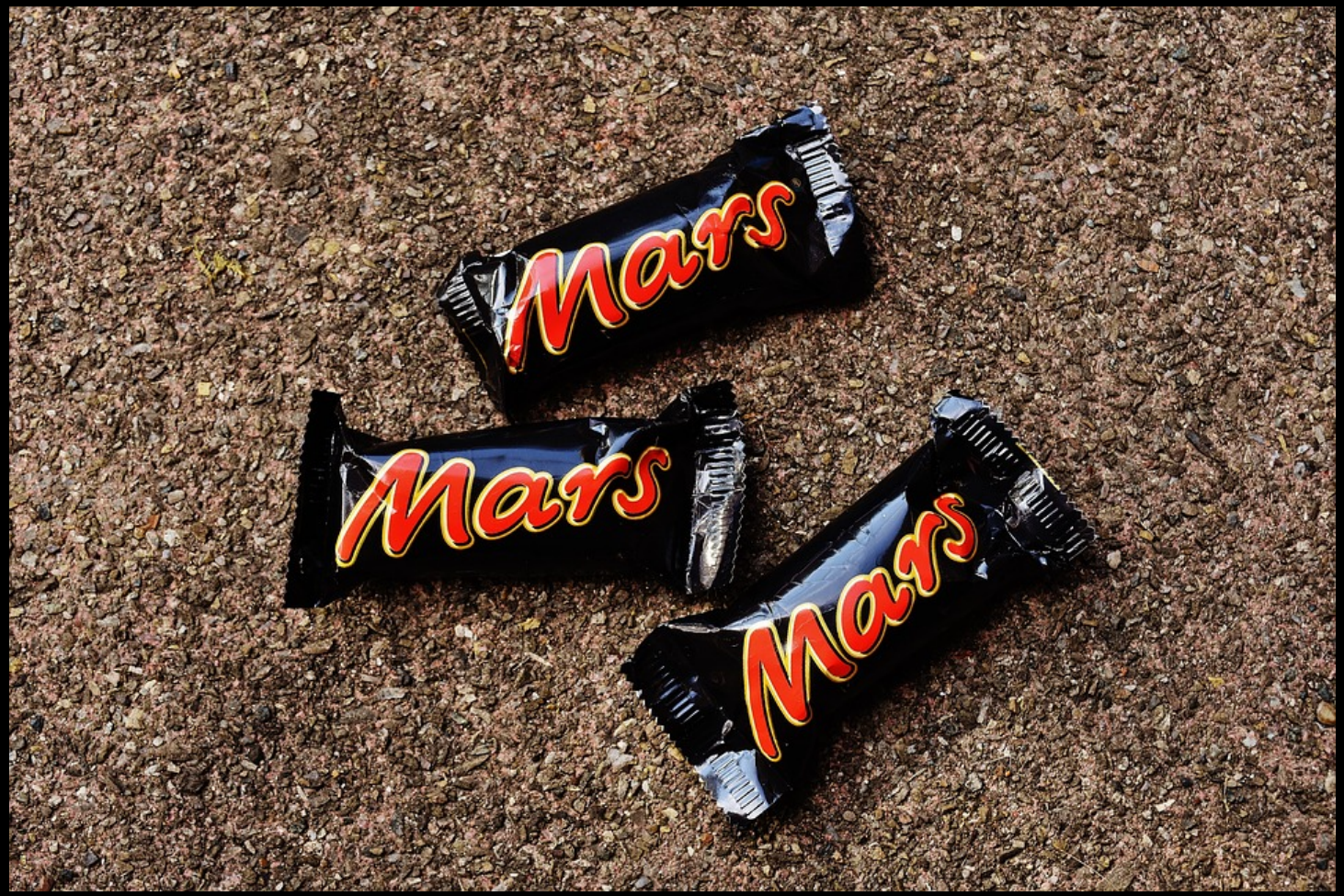


TechAtBloomberg.com

Bloomberg

Engineering

Extra content!



TechAtBloomberg.com

Bloomberg

Engineering

sentinels

```
def predict_department_expenses(budget):
    ...
    email_sender.send_email(budget)
    ...
```

```
from unittest.mock import sentinel
@unittest.mock.patch('module.to.test.email_sender', autospec=True)
def test_case(mock):
    predict_department_expenses("A LOT OF MONEY")
    mock.assert_called_with("A LOT OF MONEY")
```

sentinels

```
def predict_department_expenses(budget):
    ...
    email_sender.send_email(budget)
    ...
```

```
from unittest.mock import sentinel
@unittest.mock.patch('module.to.test.email_sender', autospec=True)
def test_case(mock):
    predict_department_expenses(sentinel.budget)
    mock.assert_called_with(sentinel.budget)
```

How does patch work?

- Temporary replace your object with another object
- By default: a new MagicMock
- Just a setattr

Attr to set

```
'package.module.target'
```

Path to set

How does patch work?

Attr to set

```
'package.module.target'
```

Path to set

```
#package/module.py
target = MagicMock()
```

How does patch work?

Attr to set

```
'package.module.target'
```

Path to set

```
# myfile.py
from package.module import target
print(target)
```

```
#package/module.py
target = MagicMock()
```

How does patch work?

Attr to set

```
'myfile.target'
```

Path to set

```
# myfile.py
from package.module import MagicMock()
print(target)
```

```
#package/module.py
target = real_object
```

Mock fixtures

```
import os
from unittest.mock import patch
import pytest

@pytest.fixture
def os_system_mock():
    with patch('os.system') as os_system:
        os_system.return_value = 1
        yield os_system

def test_calling_os(os_system_mock):
    os.system("echo 1")
```

Naming your mocks

```
def printer(a, b):
    print(a if unknown_variable else b)
printer(Mock(), Mock())


# <Mock id='140469819538008'>
```

Naming your mocks

```
>>> Mock(name="mymock")
<Mock name='mymock' ='139811286151008'>
```

```
>>> Mock(name="mymock").many.chained().calls
<Mock name='mymock.many.chained().calls' ='139811274838536'>
```

Kahoot!

Go to https://kahoot.it/

CODE: change screen!

Links

- https://docs.python.org/3/library/unittest.mock.html
- http://xunitpatterns.com/Test%20Double.html
- https://martinfowler.com/bliki/TestDouble.html
- https://pyvideo.org/pycon-us-2018/demystifying-the-patch-function.html
- https://github.com/python/cpython/blob/master/Lib/unittest/mock.py

Questions?

Thanks

TechAtBloomberg.com

Bloomberg

Engineering