Stub

Mock

Patch

Dance with shadows

María Andrea Vignau

Dance with shadows

- Why use mockers?
- How to use **patch**.
- Simulators: stub y mock.
- Libraries used for web development

Why testing

Automated testing

How to use fakers

Introduction

Introduction

- Normaly, code don’t work as you first typed.
- So, everybody need to test
- As the code grows, there is more work to maintain, modify and retest

Introduction

Tests automation types

- Integration test: a whole feature.
- Unit tests: specific functions.

Mockers - Use cases

- Testing piece by piece the software
- Using external APIs or special hardware.
- Input/Output operations
- Improbable events

Mockers

- Used instead of functions, classes, objects, libraries
- Predictable, Repeatable, and Fast
- Lightweight and cheap

Common Errors

Scope

Tradeoffs

Inverse dependency

Patching

Patching – Common errors

- YES

  Patching the imported object.

- NO

  Patching the origin from where it was imported.

Patching – Common errors

When used in b.py

```
from a import Class
```

```
@patch("b.Class")
```

```
import a
```

```
@patch("a.Class")
```

Decorator

Context manager

Manually

Patching - Scopes

Patching - Decorator

```
@patch('module.ClassB')
@patch('module.functionA')
def test_some_func(self, mock_A, mock_B):
```

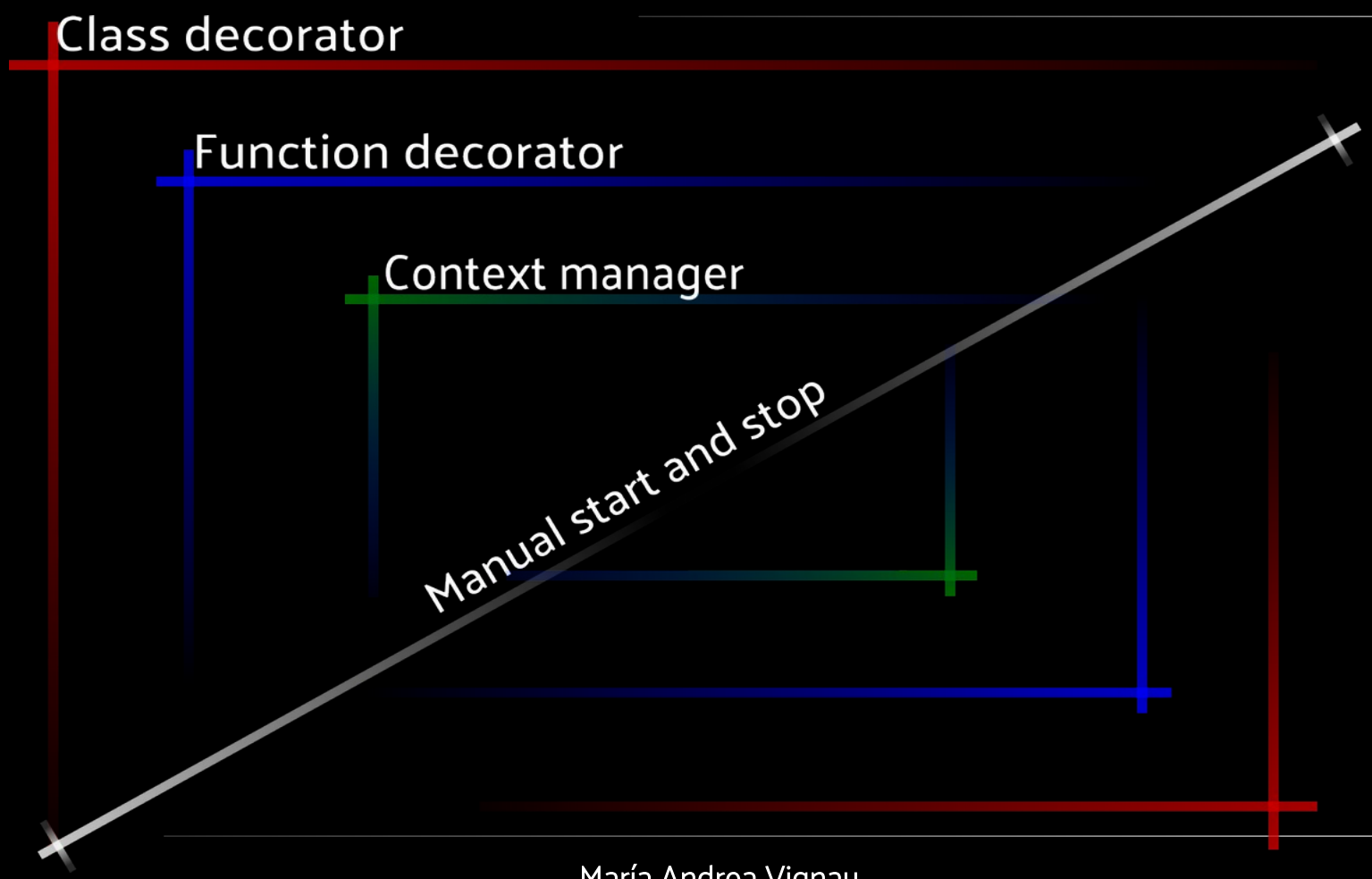
Patching – Context Manager

```
import client
with patch('client.stats') as mock:
    mock.return_value=’some’
    response = client.get('/stats/')
```

Patching - Manually

```
patcher = patch('package.module.ClassName')
from package import module
original = module.ClassName
new_mock = patcher.start()
patcher.stop()
```

Patching - Scopes

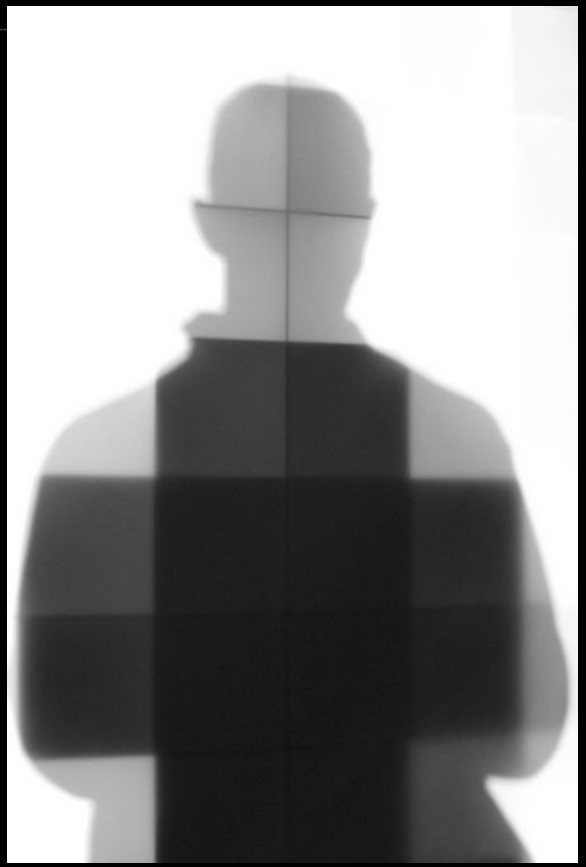


Patching – Tradeoffs

- Strong bound to the code to be tested
- Introducing behavioral changes in run time.
- Complex and expensive

Concept

Example



Dependency injection

Dependency injection

- Inject the objects, classes, etc needed by a piece of code as arguments.
- This allow us direct replacements.

Stub

Small methods used

instead of real code

```
class StubAnimal:
  def legs(self):
    return 4
```

Dependency injection

```
from A import Animal

class Dog(Animal):

    def __init__(self):


pet = Dog()
```

```
from A import Animal

class Dog():

    def __init__(self, class):


pet = Dog(class=Animal)
```

Dependency injection

Using a stub object

```
from B import Perro
class StubAnimal:
    def legs(self):
        return 4

pet = Dog(class=StubAnimal)
```

MagicMock vs Mock

Return value

Side Effect

Specs

Simulators: Mocks

Mocks – MagicMock

- MagickMock is a Mock object to which the magical methods has been added.

Mocks – Return Value

- The Mock's return value can be specified.

```
> mock_fn.return_value = "desired value"
> mock_fn()
"desired value"
```

Mocks – Side effect

- Can be used to return exceptions or multiple values.

```
> mock.side_effect = [5, 4, 3, 2, 1]
> mock(), mock(), mock()
(5, 4, 3)
> mock.side_effect = Exception('Boom!')
```

Mocks – Specs

- Mocks, by design, will NEVER return an error if we use an attribute which doesn't exists..

It will be CREATED.

- This can make us fall in a **fake security trap**

Mocks – Specs

- **Spec** copy attributes from a class in a Mock, y will return an error when trying to access a missing one.
- **Autospec** will create specs of internal objects, in a recursively way.

Mocks – Specs

```
> def function(a, b, c): pass
> mock_function = create_autospec(function,
return_value='all fine')
> mock_function(1, 2, 3)
'all fine'
> mock_function('wrong arguments')
TypeError: <lambda>() takes exactly 3 arguments
(1 given)
```

Mocks – Wrap (or spy)

- When a function is wrapped, it is really called.
- But the wrapping mock registers every call's arguments

Only calls

Check arguments

Mock - Assertions

Mock - Assertions

`assert_called`: It was call?

`assert_called_once`: It was call once?

`assert_not_called`: It was never call?

Mock - Assertions

`assert_called_with:` Called in that specific way.

`assert_called_once_with:` Called only once in this way.

`assert_any_call:` Was it call in this way?

`assert_has_calls:` Is this the exact list of calls and their args?.

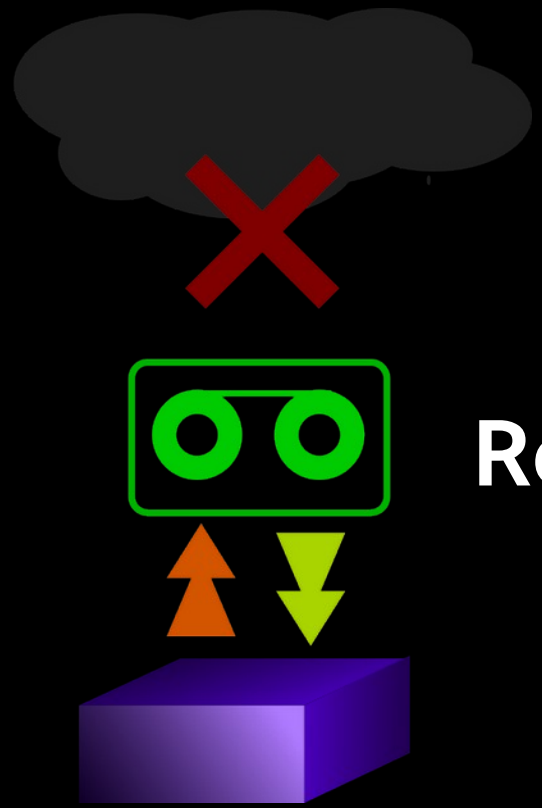
VCR

Moto

Fakers for web developer

VCR – A recorder

Saving



Reading

VCR – A recorder

- Why? Speed and Continuous Integration
- Record interactions with remote server

```
@pytest.mark.vcr()
def test_one():
    response = urlopen('http://www.one.org').read()
```

Moto – Faking AWS Boto

```
import boto3

class MyModel(object):

    def save(self):

        s3 = boto3.client('s3',...)

        s3.put_object(...)
```

```
from moto import mock_s3

from mymodule import MyModel

@mock_s3

def test_my_model_save():

    model_instance = MyModel()

    model_instance.save()
```

Dance with shadows

- Why use **mockers**?
- How to use **patch**.
- Simulators: stub y mock.
- Libraries used for web development

Contact

mavignau

marian-vignau

Dance with shadows