Mocking in Tests

July 16 2020

Rahul C

bitsapien

github.com/bitsapien

twitter.com/bitsapien_log

Senior Consultant

ThoughtWorks®

Senior Consultant

ThoughtWorks®

Rahul C

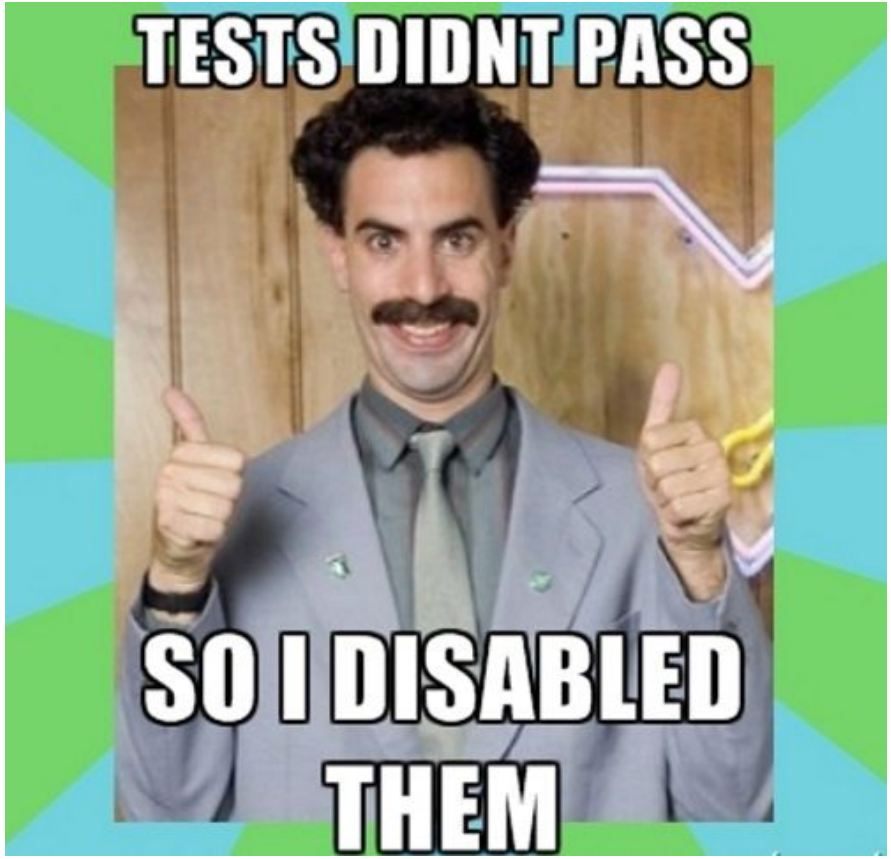
bitsapien

github.com/bitsapien

twitter.com/bitsapien_log

When do you say you are confident about the code you pushed on a Friday ?

What happens when your tests fail?


TESTS DIDNT PASS
SO I DISABLED THEM

😁 Tests are the reason developers sleep well at night

We write all kinds of tests around our software

Unit Tests | All other kind of tests

How are **unit tests** written?

 + soil + fertilisers = 

```
public class OrderStateTester extends TestCase {
  private static String TALISKER = "Talisker";
  private static String HIGHLAND_PARK = "Highland Park";
  private Warehouse warehouse = new WarehouseImpl();

  protected void setUp() throws Exception {
    warehouse.add(TALISKER, 50);
    warehouse.add(HIGHLAND_PARK, 25);
  }
  public void testOrderIsFilledIfEnoughInWarehouse() {
    Order order = new Order(TALISKER, 50);
    order.fill(warehouse);
    assertTrue(order.isFilled());
    assertEquals(0, warehouse.getInventory(TALISKER));
  }
  public void testOrderDoesNotRemoveIfNotEnough() {
    Order order = new Order(TALISKER, 51);
    order.fill(warehouse);
    assertFalse(order.isFilled());
    assertEquals(50, warehouse.getInventory(TALISKER));
  }
```

Classicist’s way of testing

```
public class OrderInteractionTester extends MockObjectTestCase {
  private static String TALISKER = "Talisker";

  public void testFillingRemovesInventoryIfInStock() {
    //setup - data
    Order order = new Order(TALISKER, 50);
    Mock warehouseMock = new Mock(Warehouse.class);

    //setup - expectations
    warehouseMock.expects(once()).method("hasInventory")
      .with(eq(TALISKER),eq(50))
      .will(returnValue(true));
    warehouseMock.expects(once()).method("remove")
      .with(eq(TALISKER), eq(50))
      .after("hasInventory");

    //exercise
    order.fill((Warehouse) warehouseMock.proxy());

    //verify
    warehouseMock.verify();
    assertTrue(order.isFilled());
  }
```

Mockist’s way of testing

What is a mock ?

Dummy Objects

Fakes

Stubs

```
//setup - expectations
warehouseMock
  .with(eq(TALISKER),eq(50))
  .will(returnValue(true));
```

Spies

```
//setup - expectations
warehouseMock.expects(once()).method("hasInventory")
```

Another example

```
public interface MailService {
  public void send (Message msg);
}
public class MailServiceStub implements MailService {
  private List<Message> messages = new ArrayList<Message>();
  public void send (Message msg) {
    messages.add(msg);
  }
  public int numberSent() {
    return messages.size();
  }
}
```

class OrderStateTester...

```
  public void testOrderSendsMailIfUnfilled() {
    Order order = new Order(TALISKER, 51);
    MailServiceStub mailer = new MailServiceStub();
    order.setMailer(mailer);
    order.fill(warehouse);
    assertEquals(1, mailer.numberSent());
  }
```

Classicist

State Verification

class OrderInteractionTester...

```
  public void testOrderSendsMailIfUnfilled() {
    Order order = new Order(TALISKER, 51);
    Mock warehouse = mock(Warehouse.class);
    Mock mailer = mock(MailService.class);
    order.setMailer((MailService) mailer.proxy());

    mailer.expects(once()).method("send");
    warehouse.expects(once()).method("hasInventory")
      .withAnyArguments()
      .will(returnValue(false));

    order.fill((Warehouse) warehouse.proxy());
  }
}
```

Mockist

Behaviour & State Verification

Open Box Testing

Closed Box Testing

Classicist or Mockist way of testing

Mocks bring in a lot of power!


WITH
GREAT POWER COMES
GREAT RESPONSIBILITY

Classicist or Mockist way of testing

How does it impact your test/code design?

Test Setup Phase

Test Isolation

Test Isolation

993 out of 1293 tests failing

1 out of 1293 tests failing

Finding the root cause in test report

Lost opportunity to detect coupling

Behaviour verification

Behaviour verification

```
// ARRANGE
when(batchCalculationFileService.read(Mockito.any(String.class))).thenReturn(calculations);
ArgumentCaptor<CalculationModel> calculationModelCaptor = ArgumentCaptor.forClass(CalculationM
```

How does testing style impact your system’s design?

And eventually your development speed.

Example: Trust on Outside World Interface

```
import boto3

class AarogyaSetu(object):
    def __init__(self, name, value):
        self.name = name
        self.adhaar_number = adhaar_number

    def save(self, bucket_name):
        s3 = boto3.client('s3', region_name='us-east-1')
        s3.put_object(Bucket='mybucket', Key=self.name,
Body=self.adhaar_number)
```

```python
import boto3
from moto import mock_s3
from mymodule import MyModel


@mock_s3
def test_aarogya_setu_save():
    conn = boto3.resource('s3', region_name='us-east-1')
    # We need to create the bucket since this is all in Moto's
'virtual' AWS account
    conn.create_bucket(Bucket='mybucket')

    model_instance = AarogyaSetu('fsociety', 'xxxx-xxxxx-xxxxx-xxxx')
    model_instance.save()

    body = conn.Object('mybucket', 'fsociety').get()
['Body'].read().decode("utf-8")

    assert body == 'xxxx-xxxxx-xxxxx-xxxx'
```

Example: Logic coupled with side-effects

```
// users.js
import axios from 'axios';

class Users {
  static all() {
    return axios.get('/users.json').then(resp => {
      return resp.data.filter((user) => user.isActive)
     });
  }
}

export default Users;
```

```
// users.test.js
import axios from 'axios';
import Users from './users';

jest.mock('axios');

test('should fetch active users', () => {
  const users = [{name: 'Bob', isActive: true}, {name: 'Alice',
isActive: false}];
  const resp = {data: users};
  axios.get.mockResolvedValue(resp);

  // or you could use the following depending on your use case:
  // axios.get.mockImplementation(() => Promise.resolve(resp))

  return Users.all().then(data => expect(data).toEqual([{name: 'Bob',
isActive: true }]));
});
```

What if we resist the urge to mock here?

```
   // users.js
import axios from 'axios';

class Users {
  static all() {
    return axios.get('/users.json').then(resp => {
      return resp.data
     });
  }
  static filterActive(users) {
    return users.filter((user) => user.isActive)
  }
}
```

Side Effect pushed to the boundary

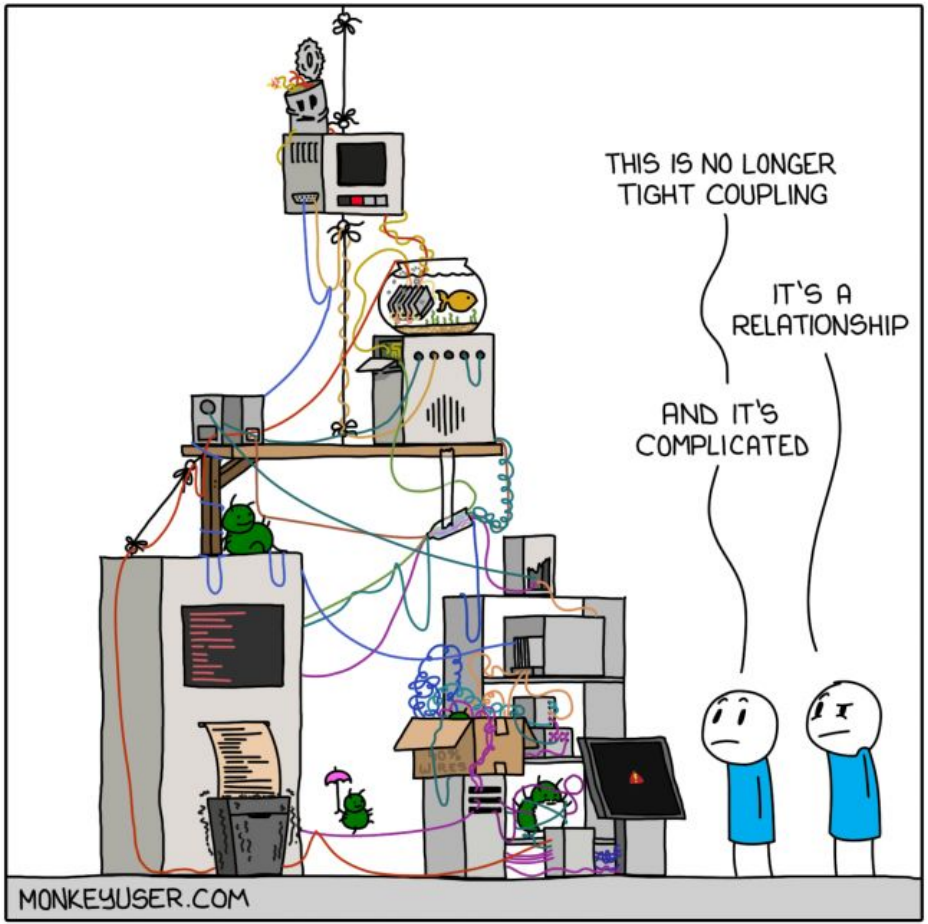
```
   // users.test.js
import axios from 'axios';
import Users from './users';

test('should filter active users', () => {
  // Given
  const users = [{name: 'Bob', isActive: true}, {name: 'Alice',
isActive: false}];
  // When
  const activeUsers = Users.filterActive(users)
  // Then
  expect(activeUsers).toEqual([{name: 'Bob', isActive: true}])
```

Coupling is bad.

Coupling is very bad!

NEXT LEVEL


THIS IS NO LONGER TIGHT COUPLING
IT'S A RELATIONSHIP
AND IT'S COMPLICATED
MONKEYUSER.COM

How to detect coupling ?

… and ofcourse don’t try to mock your way out.

Using composition

Example: Test an express app

```
const express = require('express');
const app = express();


app.get('/', function (req, res) {
  res.send('Hello World!')
});


app.listen(3000, function () {
  console.log('Example app listening on port 3000!')
});
```

```
const express = require('express');


const hello = require('./hello.js');
const handleListen = require('./handleListen');
const log = require('./log');


const port = 3000;
const app = express();


app.get('/', hello);


app.listen(port, handleListen(port, log));
```

Mocks ain’t evil

afterall

Balancing unit tests and integration tests is important

QAs and devs need to come together
to make the world a better(*aka tested*)
place to live in

Principles to use when mocking in your tests

Fight your urge to mock in unit tests

Fight your urge to mock in unit tests

Do not use off the shelf mocking tools

Fight your urge to mock in unit tests

Do not use off the shelf mocking tools

Use test versions of external systems in integration tests.

Fight your urge to mock in unit tests

Do not use off the shelf mocking tools

Use test versions of external systems in integration tests.

Push side effects to the boundaries of your system

Fight your urge to mock in unit tests

Do not use off the shelf mocking tools

Use test versions of external systems in integration tests.

Push side effects to the boundaries of your system

Don’t take unit test coverage as the holy grail.

Fight your urge to mock in unit tests

Do not use off the shelf mocking tools

Use test versions of external systems in integration tests.

Push side effects to the boundaries of your system

Don’t take unit test coverage as the holy grail.

Test “state” not “behaviour”

Fight your urge to mock in unit tests

Do not use off the shelf mocking tools

Use test versions of external systems in integration tests.

Push side effects to the boundaries of your system

Don’t take unit test coverage as the holy grail.

Test “state” not “behaviour”

Keep it simple !

“Simple ain’t easy”

Write tests to build confidence
And
NOT to make them run fast.