INTRODUCTION TO BDD

BEHAVIOR DRIVEN DEVELOPMENT


SUB-MARINER
MARVEL COMICS GROUP
20¢
50
JUNE
02455
EXTRA-SPECIAL
50TH ISSUE!
SUB-MARINER
FIGHTING FOR HIS LIFE!
MYSTERY! DANGER! SUSPENSE! AS NAMOR ASKS:
WHO AM I ?

`whoami`

Juan Ignacio Rodríguez de león

`@jileon` en Twitter

Backend developer en OctopusLabs

WE ARE HIRING

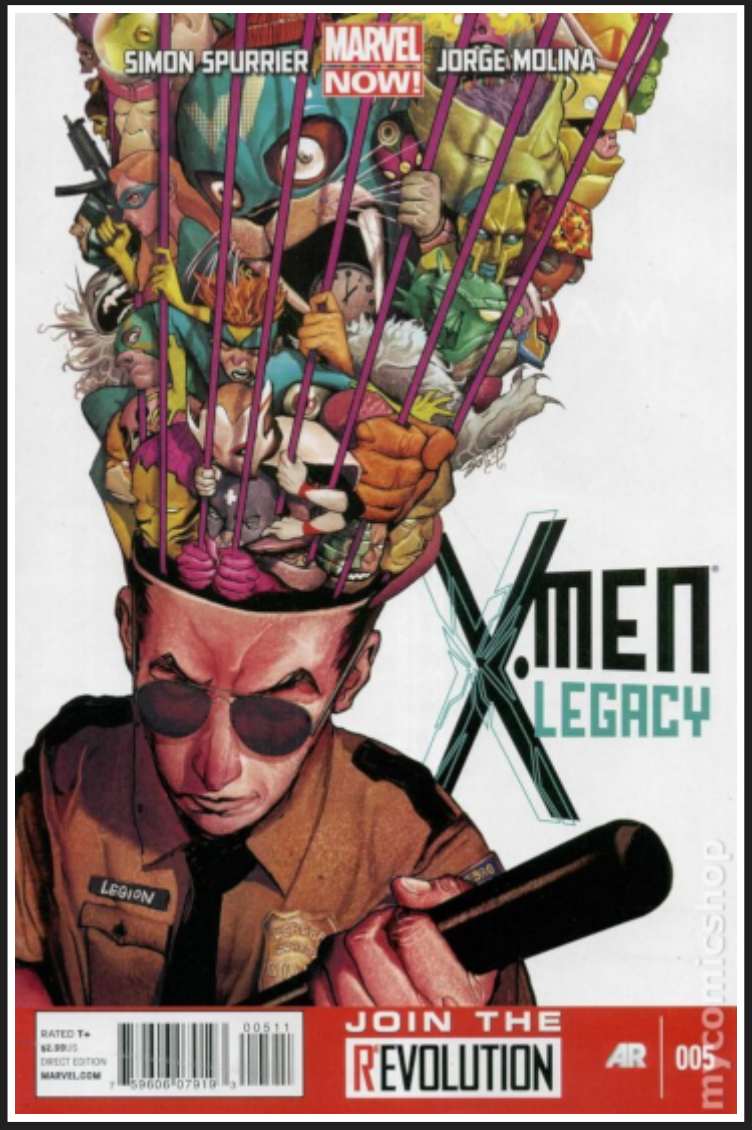
More info at:

https://octopuslabs.com/careers

HOW IS THIS TALK STRUCTURED?

- Presentation (Finished!)
- Teory
- Practice
- Libraries, tools, recomendations...

THEORY


SIMON SPURRIER
MARVEL NOW!
JORGE MOLINA
X-MEN
LEGACY
LEGION
RATED T+
$2.99 US
DIRECT EDITION
MARVEL.COM
00511
JOIN THE
REVOLUTION
AR
005
mycomicshop

In theory, there is no difference between theory and practice. In practice, there is.

--

Benjamin Brewster

WHAT IS BDD

As stated by wikipedia:

... an extension of TDD that makes use of a simple, domain-specific scripting language structured like natural language statements to ensures all development projects remain focused on delivering what the business actually needs while meeting all requirements.

`man`

WATH IS BDD

As stated by wikipedia:

~~... an extension of TDD that makes use of a simple, domain-specific scripting language structured like natural language statements to ensures all development projects remain focused on delivering what the business actually needs while meeting all requirements.~~

TLDR

- They are tests. As in TDD, we use test to drive **the design** of the software
- Usually, on a **higher level** than TDD
  Written in a **natural lenguage** , like English

Example

TLDR

- They are tests. As in TDD, we use test to drive **the design** of the software
- Usually, on a **higher level** than TDD
  Written in a **natural lenguage** , like English

Example

TLDR

- They are tests. As in TDD, we use test to drive **the design** of the software
- Usually, on a **higher level** than TDD
  Written in a **natural lenguage** , like English

Example

TLDR

- They are tests. As in TDD, we use test to drive **the design** of the software
- Usually, on a **higher level** than TDD
  Written in a **natural lenguage** , like English

Example

TARGET: SHARE KNOWLEDGE

A natural lenguaje:

- Is used **for all of us**
- **Everyone** can define, debate o improve a test
- Allow all the team to **discuss on tipically conflictive topics**: edge cases, exceptions...

TARGET: SHARE KNOWLEDGE

A natural lenguaje:

- Is used **for all of us**
- **Everyone** can define, debate o improve a test
- Allow all the team to **discuss on tipically conflictive topics**: edge cases, exceptions...

TARGET: SHARE KNOWLEDGE

A natural lenguaje:

- Is used **for all of us**
- **Everyone** can define, debate o improve a test
- Allow all the team to **discuss on tipically conflictive topics**: edge cases, exceptions...

TARGET: SHARE KNOWLEDGE

A natural lenguaje:

- Is used **for all of us**
- **Everyone** can define, debate o improve a test
- Allow all the team to **discuss on tipically conflictive topics**: edge cases, exceptions...

EXECUTABLE SPECIFICATIONS

Using this test as specifications for our software, we get specifications **written on a plain english, but executables**.

EXECUTABLE SPECIFICATIONS

Using this test as specifications for our software, we get specifications **written on a plain english, but executables**.

HOW CAN YOU "EXECUTE" ENGLISH?

Transform a plain english specification in a test

Example

NOW WE HAVE TESTS!

- But, obviously, we need to implement the code. No black magic
- Before that, lets review a few details about files and organization.

GHERKIN

The format used in this specification is called **Gherkin**

There are just a few words that have a special meaning: **`Feature`**, **`Scenario`**, **`Given`**, **`When`** y **`Then`**.

(Also **`And`**, **`But`**)

FEATURE

- Every *feature* is a capacity of our product. Each feature must have its own `.feature` file
- The keyword **Feature** must go at the begining, and gives us the opportunity to descrive the feature in one line
- The rest of text between the feature line and the first scenario can be used to describe the feature with more detail

SCENARIO

- Usually, a feature has more than one scenario, because we are interested on testing diferent aspects: changing conditiones, edge cases, different inputs, etc...
- Every diferent case is defined with the keyboard **Scenario**
- As a rule, every scenario is a complete and concrete example of a feature

GIVEN

- **Given** put our system in a known and controled state, before the first intereaction.
- (No user interactions if this part)

WHEN

- **When** This is where we describe the actions the user or external system wants to do. This must yield some changes in the system.
- Or maybe not

THEN

- **Then** This is where we check that the outcomes of our system are the expected

AND Y BUT

And y **But** are alternative versions of Given and Then

just to get a better feeling of proper english.

So, instead of writing:

```
Given I have 10000€ in my current acount
Given I have 300€ in my saving accout
```

We can write:

```
Given I have 10000€ in my current acount
And I have 300€ in my saving accout
```

GO BACK TO PRACTICE

LETS IMPLEMENT THE TEST

Example

A FEW INTERESTING DETAILS

ON STEPS

- The name of the function used to implemente the steps **it's not important!**
- Behave decide what funciones to call, and when, based only in the text of decorator
- Main idea is having **reusable steps**

A FEW INTERESTING DETAILS

ON STEPS

- The name of the function used to implemente the steps **it's not important!**
- Behave decide what funciones to call, and when, based only in the text of decorator
- Main idea is having **reusable steps**

A FEW INTERESTING DETAILS

ON STEPS

- The name of the function used to implemente the steps **it's not important!**
- Behave decide what funciones to call, and when, based only in the text of decorator
- Main idea is having **reusable steps**

REQUEST

- Every step has a common first parameter called `request`, that could be used for Sharing data
- This data can be shared at a feaure level or a scenario level

REQUEST

- Every step has a common first parameter called `request`, that could be used for Sharing data
- This data can be shared at a feaure level or a scenario level

PARAMETER CAPTURE

- One interesting ability is to "capture" some part of the text of the step
- This allow us to have parametrizable, and hence, more reusable steps

Lets see this with another example

DIFFERENCES WITH TDD

- Emphasis is not in the code, but in the **functionality**
- Specification is written in natural language (i.e. **english**)
- Spoken by **all the members** of the team: Developers testers, Q/A, product owners, skateholders, users...

DIFFERENCES WITH TDD

- Emphasis is not in the code, but in the **functionality**
- Specification is written in natural language (i.e. **english**)
- Spoken by **all the members** of the team: Developers testers, Q/A, product owners, skateholders, users...

DIFFERENCES WITH TDD

- Emphasis is not in the code, but in the **functionality**
- Specification is written in natural language (i.e. **english**)
- Spoken by **all the members** of the team: Developers testers, Q/A, product owners, skateholders, users...

DIFFERENCES WITH TDD

- Emphasis is not in the code, but in the **functionality**
- Specification is written in natural language (i.e. **english**)
- Spoken by **all the members** of the team: Developers testers, Q/A, product owners, skateholders, users...

PRACTICE


MARVEL
25
CAPTAIN AMERICA
WELL, HERE'S ANOTHER FINE MESS YOU'VE GOTTEN US INTO, HAWKEYE!
75¢
317
MAY

A FEATURE FILE EXAMPLE

```
Feature: Transfer between accounts

  Scenario: Add savings

    Given I have 10.000€ in my current acount
    And I have 300€ in my saving accout
    When I transfer 200€ from current to saving
    Then I should have 9.800€ in my current acount
    And I should have 500€ in my saving acount
```

INSTALL BEHAVE

- **Behave** is a BDD tool written in Python
- We will see other options for other languages at the end of the talk

```
pip install behave
```

FOR OUR FIRST BDD TEST...

WE ARE GOING TO NEED ...

- A **`features`** directory
- Inside:
  - a directory named **`steps`**.
  - And a file named with extension **`.feature`***

* with the content show in the previous example

FOR OUR FIRST BDD TEST...

WE ARE GOING TO NEED ...

- A **`features`** directory
- Inside:
  - a directory named **`steps`**.
  - And a file named with extension **`.feature`***

* with the content show in the previous example

FOR OUR FIRST BDD TEST...

WE ARE GOING TO NEED ...

- A **`features`** directory
- Inside:
  - a directory named **`steps`**.
  - And a file named with extension

    `.feature`*

* with the content show in the previous example

FOR OUR FIRST BDD TEST...

WE ARE GOING TO NEED ...

- A **`features`** directory
- Inside:
  - a directory named **`steps`**.
  - And a file named with extension **`.feature`***

* with the content show in the previous example

```
mkdir features
mkdir features/steps
vim features/transfer-between-accounts.feature
```

FIRST TEST WRITTEN!

AND IT IS EXECUTABLE!

To execute it, just type on the command line:

```
behave
```

```
Feature: Transfer between accounts # features/transfer-between

  Scenario: Add savings                                  # features/t
    Given I have 10000€ in my current acount             # None
    And I have 300€ in my saving accout                  # None
    When I transfer 200€ from current to saving          # None
    Then I should have 9800€ in my current acount # None
    And I should have 500€ in my saving acount           # None


Failing scenarios:
  features/transfer-between-accounts.feature:3  Add savings

0 features passed, 1 failed, 0 skipped
```

```
You can implement step definitions for undefined steps with th

@given(u'I have 10000€ in my current acount')
def step_impl(context):
    raise NotImplementedError(u'STEP: Given I have 10000€ in m

@given(u'I have 300€ in my saving accout')
def step_impl(context):
    raise NotImplementedError(u'STEP: Given I have 300€ in my

@when(u'I transfer 200€ from current to saving')
def step_impl(context):
    raise NotImplementedError(u'STEP: When I transfer 200€ fro
```

IT WORKS!

And It even gives us a scheleton of the functions we need to implement to run the tests

A SIMPLE TEST

- Let's see a first implementation
- Given thath we have no real code to test for now, use a simulation

GIVEN...

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

from behave import *

@given("I have 10000€ in my current acount")
def step_impl(context):
    context.current_account = 10000


@given("I have 300€ in my saving accout")
def step_impl(context):
    context.savings_account = 300
```

WHEN...

```
@when("I transfer 200€ from current to saving")
def step_impl(context):
    context.savings_account += 200
    context.current_account -= 200
```

THEN...

```
@then("I should have 9800€ in my current acount")
def step_impl(context):
    assert context.current_account == 9800

@then("I should have 500€ in my saving acount")
def step_impl(context):
    assert context.savings_account == 500
```

EXECUTE BEHAVE

```
Feature: Transfer between accounts # features_002/transfer-bet

  Scenario: Add savings                                  # features_0
    Given I have 10000€ in my current acount             # features_0
    And I have 300€ in my saving accout                  # features_0
    When I transfer 200€ from current to saving          # features_0
    Then I should have 9800€ in my current acount        # features_0
    And I should have 500€ in my saving acount           # features_0

1 feature passed, 0 failed, 0 skipped
1 scenario passed, 0 failed, 0 skipped
5 steps passed, 0 failed, 0 skipped, 0 undefined
Took 0m0.000s
```

A bit more Theory

PARAMETRIZABLE STEPS

- We can parse the example text strings and get information for the steps
- The library parse is used by default
- parse defines itself as *the opposite of format()*

Change definition of first step from:

```
@given("I have 10000€ in my current acount")
def step_impl(context):
    context.current_account = 10000
```

To:

```
@given("I have {amount}€ in my current acount")
def step_impl(context, amount):
    context.current_account = int(amount)
```

```
Feature: Transfer between accounts # features_003/transfer-bet

  Scenario: Add savings                                  # features_0
    Given I have 10000€ in my current acount             # features_0
    And I have 300€ in my saving accout                  # features_0
    When I transfer 200€ from current to saving          # features_0
    Then I should have 9800€ in my current acount        # features_0
    And I should have 500€ in my saving acount           # features_0

1 feature passed, 0 failed, 0 skipped
1 scenario passed, 0 failed, 0 skipped
5 steps passed, 0 failed, 0 skipped, 0 undefined
Took 0m0.001s
```

WE CAN EVEN FORGET

ABOUT THE INTEGER CAST

```
@given("I have {amount:d}€ in my current acount")
def step_impl(context, amount):
    context.current_account = amount
```

BEHAVE HAVE ALTERNATIVE PARSERS

- **parse** is used by default
- **cfparse** let us work with cardinality
- **re** to use regular expressions

BEHAVE HAVE ALTERNATIVE PARSERS

- **parse** is used by default
- **cfparse** let us work with cardinality
- **re** to use regular expressions

BEHAVE HAVE ALTERNATIVE PARSERS

- **parse** is used by default
- **cfparse** let us work with cardinality
- **re** to use regular expressions

BEHAVE HAVE ALTERNATIVE PARSERS

- **parse** is used by default
- **cfparse** let us work with cardinality
- **re** to use regular expressions

BUT BEFORE...

- Lets forget all this financial stuff nobody cares for...
- and use more interesting examples...

POKEMON

TABLE EXAMPLES

We can add data to the steps in a tabular form

```
Feature: Pokemon search

  Scenario: Find the weakest

    Given I have this pokemons
        |name     | attack  | defense |
        |psyduck  | 20      | 60      |
        |torchic  | 20      | 60      |
        |spinda   | 10      | 80      |
        |lillipup | 10      | 50      |
    When I search for the one with less defense
    Then I should get lillipup
```

WE CAN PROCESS THIS TABLE

IN THE STEP CODE

```
@given('I have this pokemons')
def step_impl(context):
    for row in context.table:
        pokemon = Pokemon(
            row['name'],
            attack=row['attack'],
            defense=row['defense'],
            )
        setattr(context, row['name'], pokemon)
```

WE CAN HAVE SHARED PRECONDITIONS

FOR ALL THE STEPS

```
Feature: Transfer between accounts

 Background:

   Given I have this pokemons
       |name     | attack  | defense |
       |psyduck  | 20      | 60      |
       |torchic  | 20      | 60      |
       |spinda   | 10      | 80      |
       |lillipup | 10      | 50      |

 Scenario: Search for the weakest
   When I search for the one with less defense
   Then I should get lillipup
```

TAGS

- **Selection** of scenarios or features
- Can select for **one or several** tags
- Can execute the ones **not** having one or more tags

TAGS

- **Selection** of scenarios or features
- Can select for **one or several** tags
- Can execute the ones **not** having one or more tags

TAGS

- **Selection** of scenarios or features
- Can select for **one or several** tags
- Can execute the ones **not** having one or more tags

TAGS

- **Selection** of scenarios or features
- Can select for **one or several** tags
- Can execute the ones **not** having one or more tags

FEATURE WITH TAGS

Asuming this feature and sample data

```
Feature: Search pokemons

 Background:

   Given I have this pokemons
       |name      | attack  | defense |
       |psyduck   | 20      | 60      |
       |torchic   | 20      | 60      |
       |trapinch  | 20      | 60      |
       |spinda    | 10      | 80      |
       |lillipup  | 10      | 50      |
```

AND THIS STEPS

```
 @fight
Scenario: Search for the weakest
  When I search for the one with less defense
  Then I should get lillipup

@fight
Scenario: Search for the stronger
  When I search for the one with more attack
  Then I should get spinda

Scenario: Search for name
  When I search for the letter t
  Then I should get torchic
   And I should get trapinch
```

EXECUTING

JUST THE FIGHT SCENARIOS

```
behave features_006 --tags=fight
```

EXECUTINGA ALL THE SCENARIOS

BUT THE FIGHT ONES

```
behave features_006 --tags=-fight
```

WORKS IN PROGRESS

- run with `–w` flag
- turns off stdout capture
- turns off logging capture
- turns off pretty output
- only runs scenarios tagged with “@wip”
- stops at the first error

TO BE HANDLE WITH CARE

RECOMENDATIONS

- Don't mix features
- One scenario to test just one case
- it's code, store it in the repo
- High level descriptions: Do not use technology terms

DANGERS

- Don't make the system **too complex**
- Don't get used to **having some tests never passsing**
- Don't start testing the easy ones. You must go for the ones that **teaches you the most**

DANGERS

- Don't make the system **too complex**
- Don't get used to **having some tests never passsing**
- Don't start testing the easy ones. You must go for the ones that **teaches you the most**

DANGERS

- Don't make the system **too complex**
- Don't get used to **having some tests never passsing**
- Don't start testing the easy ones. You must go for the ones that **teaches you the most**

DANGERS

- Don't make the system **too complex**
- Don't get used to **having some tests never passsing**
- Don't start testing the easy ones. You must go for the ones that **teaches you the most**

RULE NUMBER 1 ON ANALISYS CLUB

- Don't tell me your solution
- Instead, tell me your problem

(Rule number 2 is Everybody lies, if you are interested)

TOOLS AND OPTIONS

- Ruby - Cucumber
- Javascript - Cucumber.js
- .Net - xBehave
- Java - JBehave

THANK YOU!