Desarrollo de una *Feature*... *(A)TDD STORY*

¿Qué es TDD?

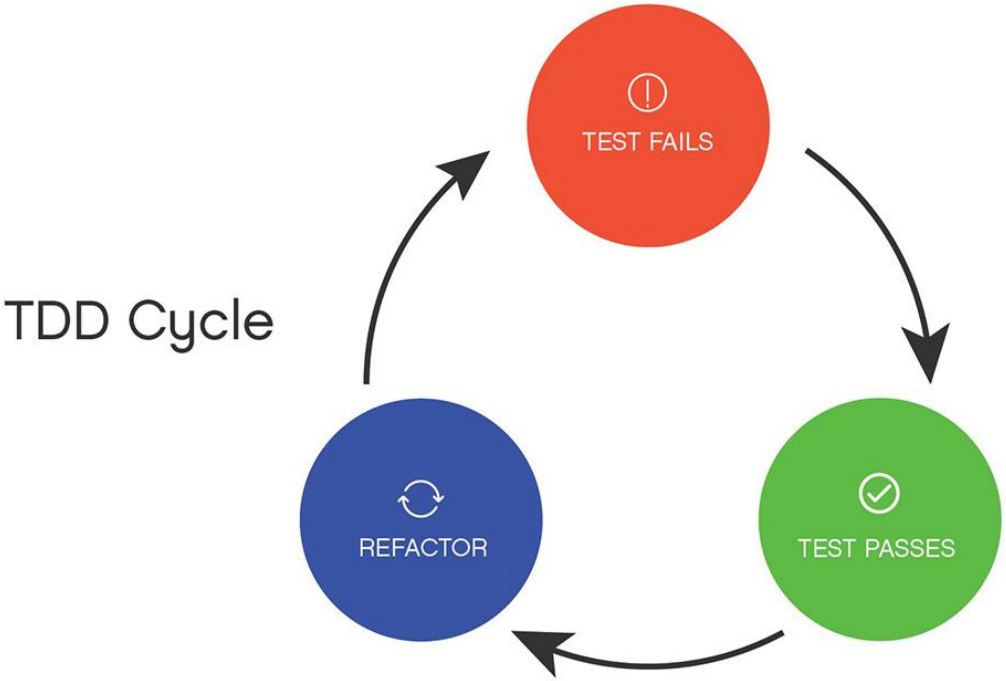
TDD

ALL CODE IS GUILTY
UNTIL PROVEN INNOCENT

Proceso Iterativo en el cual el desarrollo está GUIADO por los

TEST

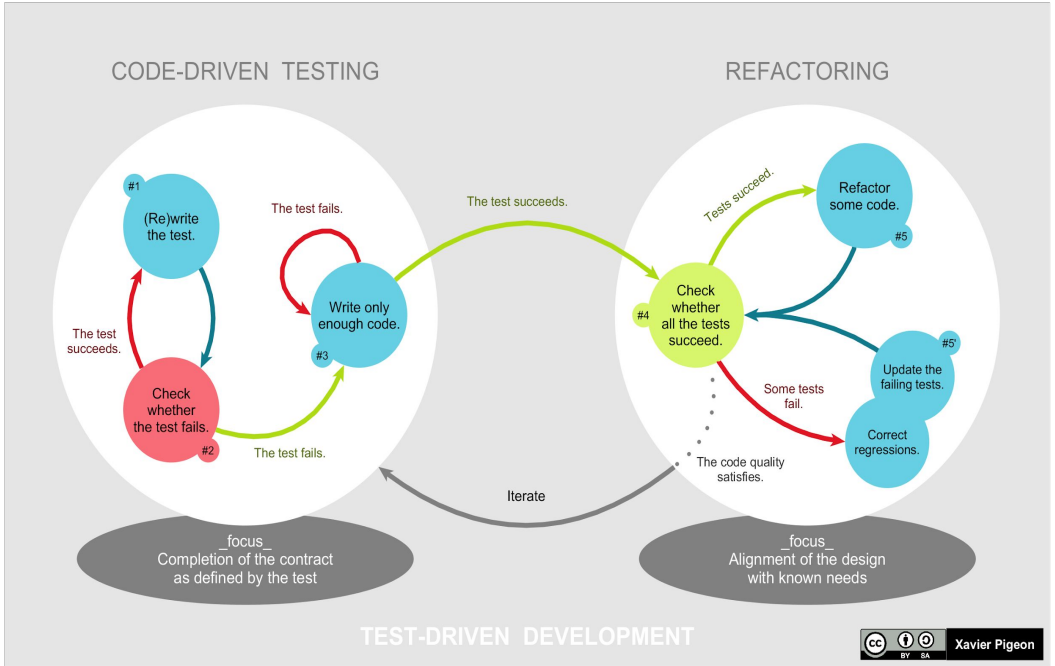
TDD Mantra



- RED: Escribe un TEST que no funcione, quizá incluso ni compile.
- GREEN: Haz que el test pase, ¡Sin importar cómo!
- REFACTOR: Elimina duplicidades y el “smelly code”

TDD y Refactorización

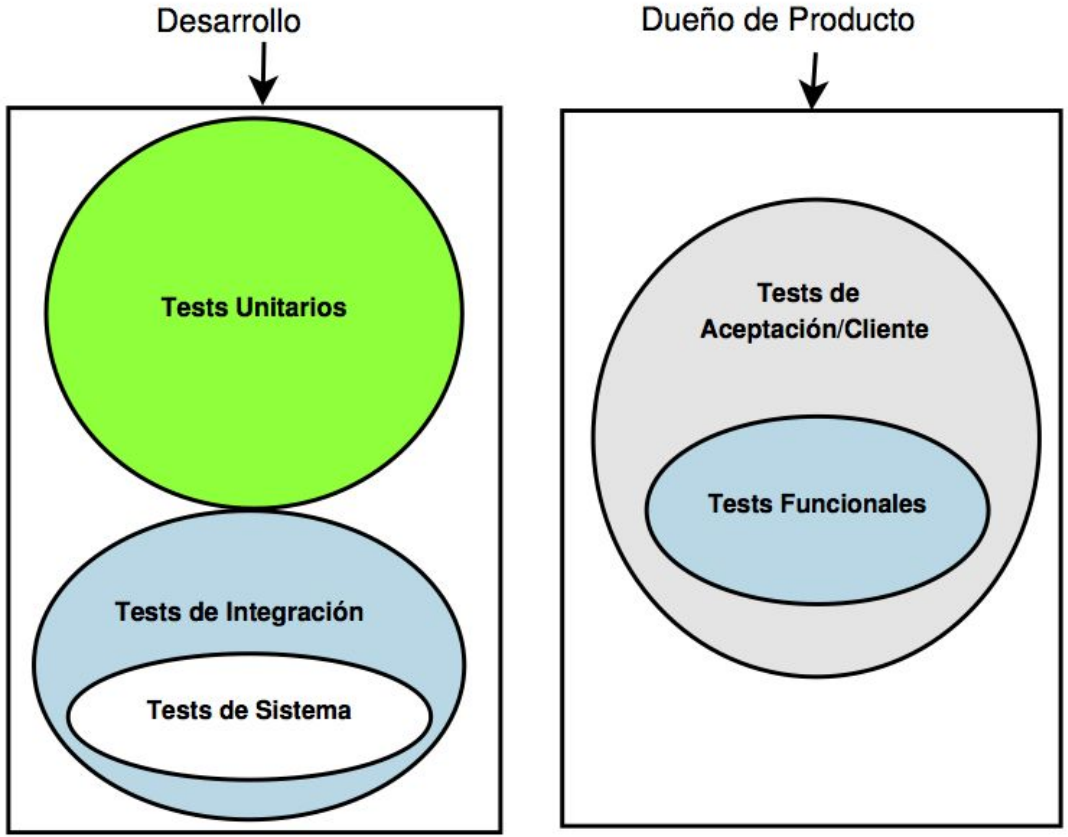
1. Añade un pequeño test
2. Corre todos los test y falla
3. Haz un pequeño cambio
4. Refactoriza para eliminar el código duplicado



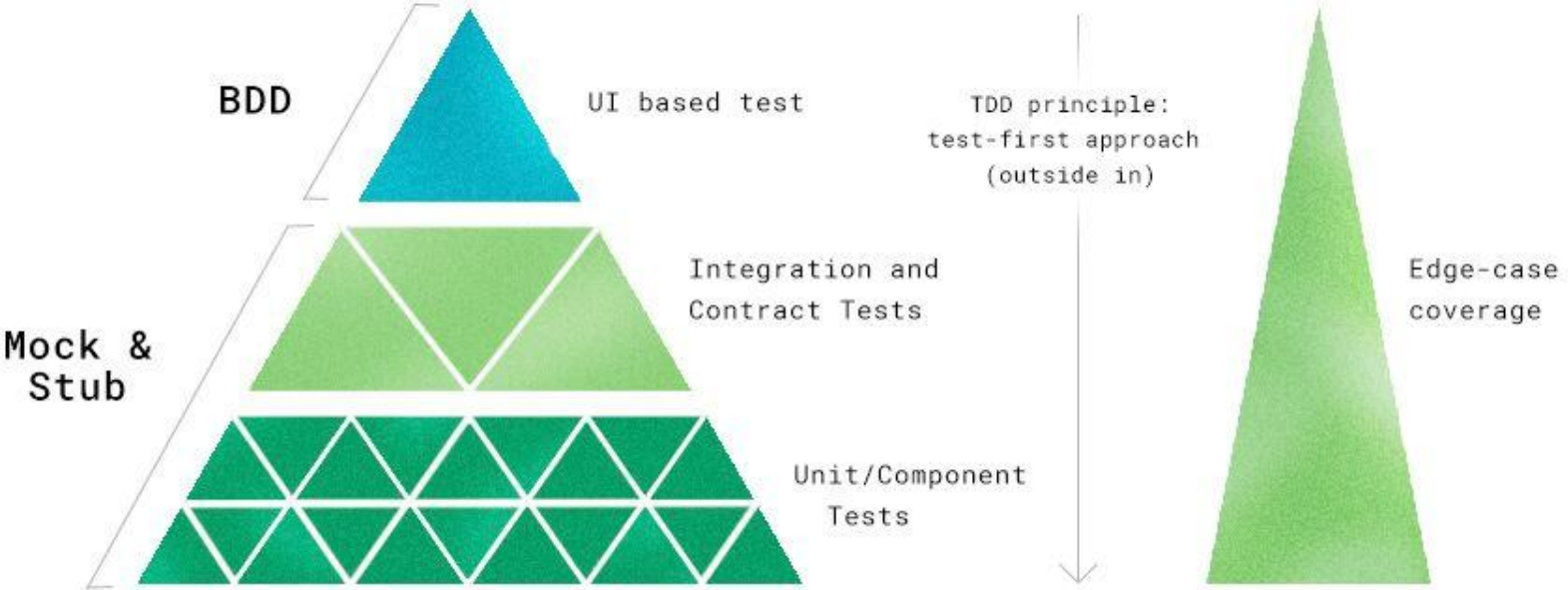
Tipos de Test (ATDD)

- ***Test Unitario:*** Test que nos ayudan en nuestro desarrollo diario deben ser ***F.I.R.S.T*** (Fast, Fast, Independent, Repeatable, Small y Transparent.)
- ***Test Integración:*** Dentro del test unitario aquel que no permite evaluar el comportamiento de nuestro desarrollo con datos reales.
- ***Test Sistema:*** Test de integración el que probamos todas la piezas de software/infraestructura que componen nuestra aplicación
- ***Test Aceptación/Funcionales:*** *Test de usuario* donde se prueba que el sistema en conjunto cumple los criterios de aceptación

*** nota: Los test de carga y rendimiento caerán de un lado u otro dependiendo del negocio en cuestión**

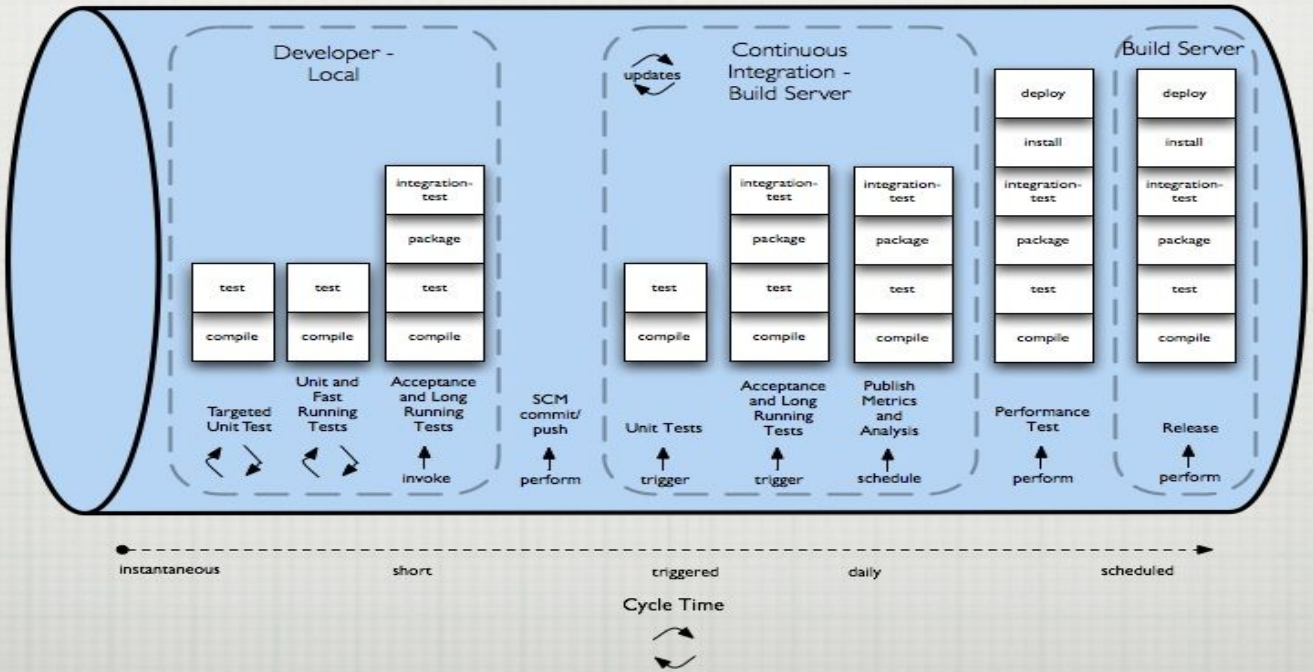


La pirámide del TDD

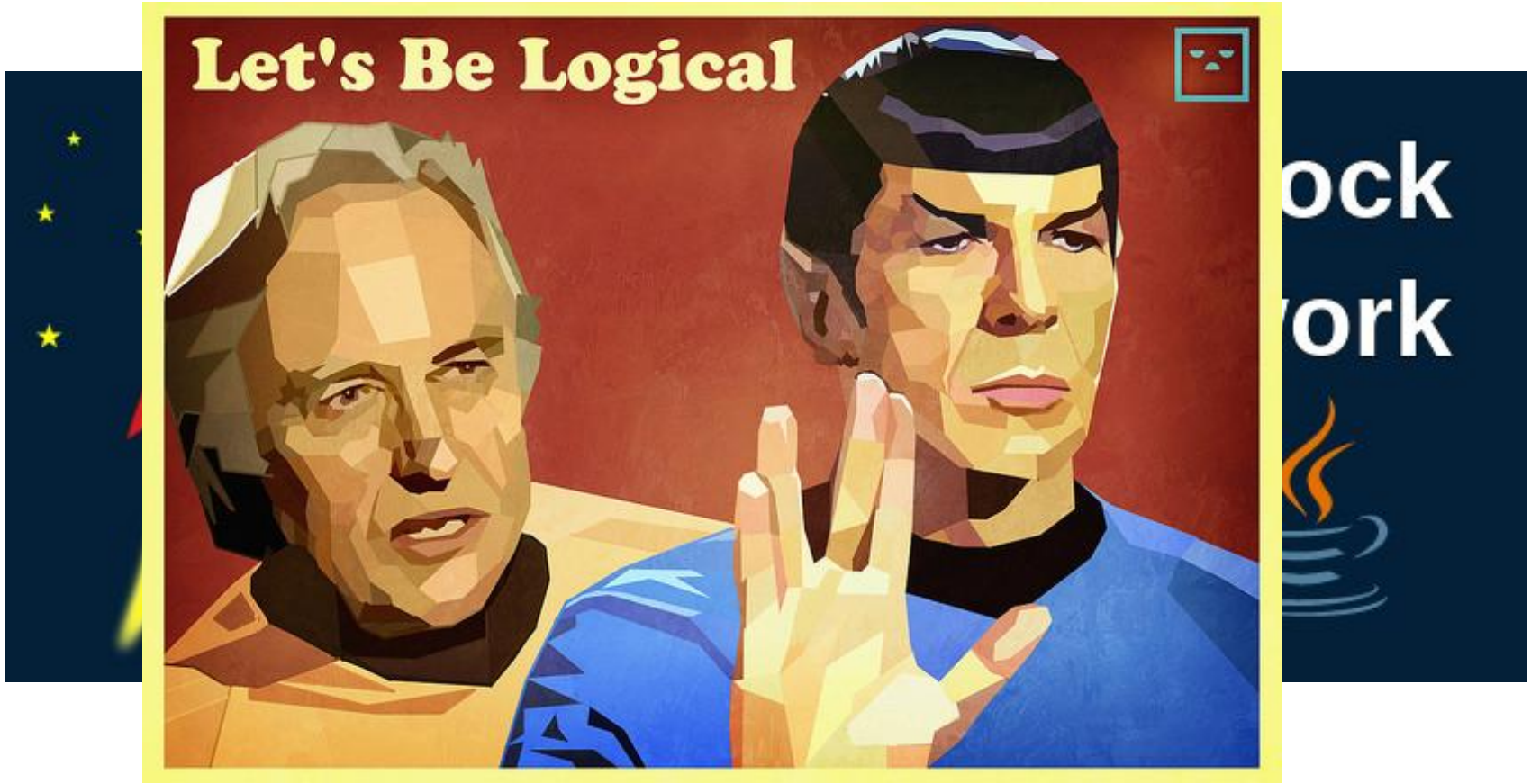


Build Pipeline

Example Build Pipeline



La Herramienta



Estructura de un Test

1. ***Setup(Given):*** Donde seteamos lo necesario para ejecución del test.
2. ***When:*** Aquí invocamos al método que queremos testear.
3. ***Then:*** Bloque donde pondremos nuestras aserciones. En Spock se representa como comparaciones booleanas.
4. **Expect:** Bloque donde podremos estimular el método a testear al a vez que hacemos un assert.
5. ***CleanUp:*** Fase de *teardown* de nuestro test. Dejamos el sistema en posición de poder realizar el resto.

```
def "two plus two should equal four"() {
    given:
        int left = 2
        int right = 2

    when:
        int result = left + right

    then:
        result == 4
}
```

```groovy
import spock.lang.Specification;

/**
* @author eloquent.developer.
*/
class PublisherSpec extends Specification {
    Publisher publisher = new Publisher()

    def setup() {
        Subscriber subscriber = Mock()
        Subscriber subscriber2 = Mock()
    }

    def "should send messages to all subscribers"() {
        given:
            publisher.subscribers << subscriber // << is a Groovy shorthand for List.add()
            publisher.subscribers << subscriber2

        when:
            publisher.send("hello")
        then:
            1 * subscriber.receive("hello") // exactly one call
            1 * subscriber2.receive("hello")
    }
}
```

Spock Killing Features

def nos permite declarar dinámicamente nuestras variables, ***inferencia de tipos!!***

Listas *Sólo tenemos que poner los valores entre brackets y esta se instancia e inicializa sola*

Para ***verificar una excepción*** solo necesitamos el método thrown(), en caso de excepción verificará el tipado de esta sin para el test.

```
def "Should get an index out of bounds when removing a
non-existent item"() {
    given:
        def list = [1, 2, 3, 4]

    when:
        list.remove(20)

    then:
        thrown(IndexOutOfBoundsException)
        list.size() == 4
}
```

```
        def list = [1, 2, 3, 4]

    when:
        list.remove(0)

    then:
        list == [2, 3, 4]
}
```

Spock Killing Features

aserciones más simples spock es capaz de comparar en profundidad tanto listas como objetos simplificando el trabajo de testing

```
Condition not satisfied:

info.lenguajes.nombre.first() == 'Java'
|    |         |      |       |
|    |         |      Groovy  false
|    |         [Groovy, Java] 5 differences (16% similarity)
|    |                        (Groo)v(y)
|    |                        (Ja--)v(a)
|    [[nombre:Groovy, conocimientos:10], [nombre:Java,
conocimientos:9]]
[nombre:Iván, lenguajes:[[nombre:Groovy, conocimientos:10],
[nombre:Java, conocimientos:9]]]

Expected :Java

Actual   :Groovy
```

Spock Killing Features

Y cuando falla…

```
def "numbers to the power of two"(int a, int b, int c)
{
  expect:
      Math.pow(a, b) == c

  where:
```

```
Condition not satisfied:

Math.pow(a, b) == c
     |   |  |  |  |
     4.0 2  2  |  1
               false

Expected :1

Actual   :4.0
```

Data Driven Testing: Spock
Data Tables para poder teste
nuestro método/clase con va
de datos

Mocks

Por asignación (groovy style): esto nos permite asignar nuestro mock aun tipo distinto

```
def paymentGateway = Mock(PaymentGateway)
```

Tipo inferido de la variable

```
PaymentGateway paymentGateway = Mock()
```

El mock retornara un ***valor por defecto sin que se haga una llamada real a la instancia de la clase mockeada.***

```
when:
    def result = paymentGateway.makePayment(12.99)

then:
    result == false
```

Stub & Verficación de Mocks

Spock también nos permite **instrumentar la respuesta** de nuestros mocks

```
paymentGateway.makePayment(_) >> true
paymentGateway.makePayment(_) >>> [true, true, false, true]
```

Para verificar si se ha llamado al mock de la forma esperada

```
def "Should verify notify was called"() {
    given:
        def notifier = Mock(Notifier)

    when:
        notifier.notify('foo')

    then:
        1 * notifier.notify('foo')
}
```

O usando wildcards para más flexibilidad

```
2 * notifier.notify(!'foo')
2 * notifier.notify(_)
```

Referencias

- Testing Java Microservices
- Testing Java with Spock
- Test-Driven-Development by Example

Wanna Play??

Triangular en TDD