GeeksHubs
academy _

TDD/BDD

Historia del sector

Problemas en el sector.

A lo largo del tiempo, el Standish Group reveló 50.000 proyectos fallidos y en 1994 se obtuvieron los siguientes resultados:

- Porcentaje de proyectos que son cancelados: 31 %
- Porcentaje de proyectos problemáticos: 53 %
- Porcentaje de proyectos exitosos: 16 % (pero estos sólo cumplieron, en promedio, con el 61 % de la funcionalidad prometida)

Sin embargo, en 2004 los resultados seguían sin ser alentadores:

- Porcentaje de proyectos exitosos: crece hasta el 29 %.
- Porcentaje de proyectos fracasados: 71 %.

Problemas en el sector.

Según el informe de Standish, las diez causas principales de los fracasos, por orden de importancia, son:

1. Escasa participación de los usuarios
2. Requerimientos y especificaciones incompletas
3. Cambios frecuentes en los requerimientos y especificaciones
4. Falta de soporte ejecutivo
5. Incompetencia tecnológica
6. Falta de recursos
7. Expectativas no realistas
8. Objetivos poco claros
9. Cronogramas irreales
10. Nuevas tecnologías

Test Driven Development (TDD)

¿Qué es TDD?

El TDD (desarrollo dirigido por tests) es una proceso de software que involucra :

- Escribir primero las pruebas, que comprueba nuestro código.
- Después escribir nuestro código fuente
- **Refactoring**
  - Revisión constante del código mejorando
    - La legibilidad
    - Eliminando duplicados,
    - Decisiones de diseño...

¿Qué es TDD?

Para cumplir con TDD se tiene que realizar código simple (Baby Steps)
Este concepto con la práctica quedará más claro.

Familiarízate con el término KISS principle

Con esta técnica se intenta conseguir código más robusto, seguro, mantenible y de calidad.

¿Qué es TDD?

Para el proyecto:

- Conseguimos código altamente reutilizable.
- El trabajo en equipo se hace más fácil, une a las personas.
- Nos permite confiar en nuestros compañeros aunque tenga menos experiencia.
- Multiplica la comunicación entre los miembros del equipo.
- Poder dormir tranquilos por la noche

¿Qué es TDD?

El Desarrollo Dirigido por Tests (Test Driven Development), al cual me referiré como TDD, es una técnica de diseño e implementación de software incluida dentro de la metodología XP

http://agilemanifesto.org/iso/es/principles.html

http://www.extremeprogramming.org/

https://openlibra.com/es/book/scrum-y-extreme-programming-para-programadores

Ciclo de Vida del TDD

EL CICLO DE VIDA TDD

El ciclo de vida de TDD se divide en estos 3 puntos fundamentales.

1. Rojo:
   a. Escribir prueba
2. Verde:
   a. Desarrollar código que pase la prueba
3. Lila(Azul):
   a. Refactoring, eliminar duplicidad y hacer mejoras

RED

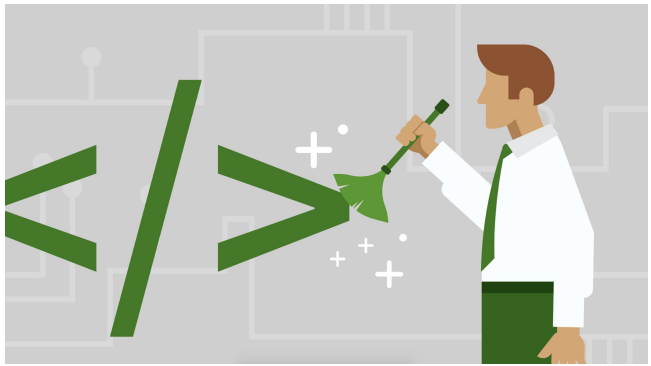
- Escribimos un test y los dejamos en rojo esto significa que el test falle.
- El test tiene que FALLAR, no dar error.

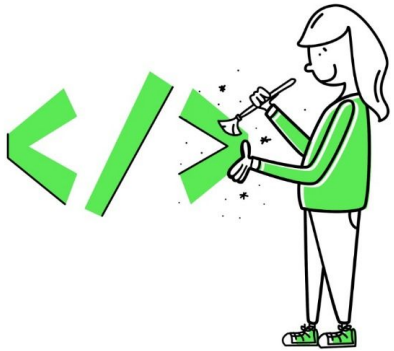
GREEN

- Escribimos la mínima cantidad de código que resuelva el test.
- Se trata de solucionar SOLO ese test. Principio KISS.
- No debemos adelantar acontecimientos o presuponer futuras necesidades.

BLUE (Refactoring)

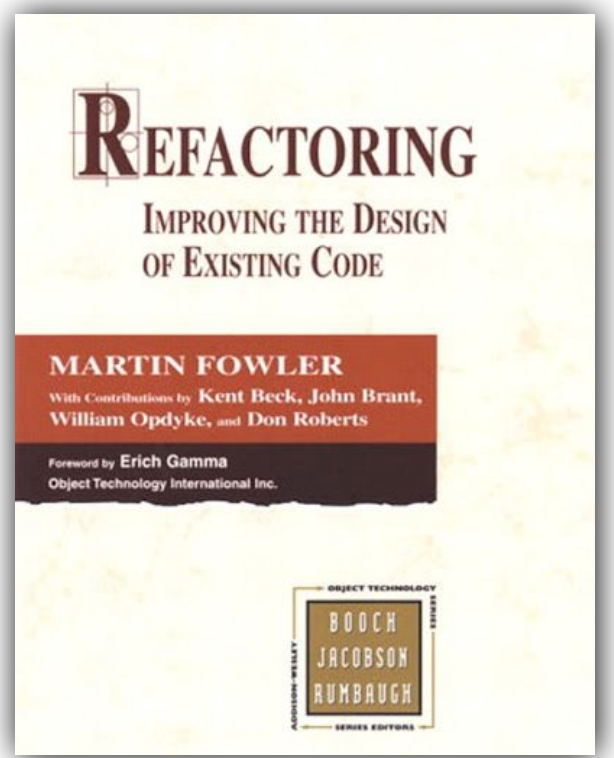
- Analizamos nuestro código y buscamos posibles mejoras.
- Renombrando variables/métodos.
- Eliminación de smells.
- Eliminación de duplicidades.





BLUE (Refactoring)

Refactorizar no significa reescribir el código; reescribir es más general que refactorizar. Segun Martín Fowler, refactorizar es *"modificar el diseño sin alterar su comportamiento"*. A ser posible, sin alterar su API pública.



Limitaciones del TDD

Limitación del TDD

Interfaces gráficas de usuario

- No existe manera fácil de testearlas.
- Hay otras metodologías quizá mejor orientadas a este tipo de testing,como el BDD.

Base de datos

- Es complicado de testear el código que accede a base de datos.
- Se necesita tener una base de datos conocida y restaurar su estado antes de cada test

Estructura de un proyecto Testing

Estructura de un proyecto Testing

Para la realización de un desarrollo guiado por pruebas 'Development Testing' se necesita tener un entorno configurado para la realización de las mismas en local.

En la práctica vamos a usar JavaScript Vanilla como lenguaje de programación, pero con Typescript. Es un superconjunto de JavaScript, que esencialmente añade tipos estáticos y objetos basados en clases.

Por encima usaremos un Framework de Testing denominado 'Jest'. Éste permite de manera muy simple hacer pequeñas pruebas para llegar a nuestro propósito de desarrollo.

Para poder usar 'Jest' vamos a tener que instalar un distribuidor de paquetes llamado 'npm'. Para poder usar 'npm' necesitamos tener instalado node-js

Node-js & npm

Para la instalación de estos recursos hemos dejado los enlaces en las referencias.
Hemos elegido unos pequeños tutoriales muy fáciles de seguir dependiendo de la plataforma que uses (Win/Lin/Mac).

Para saber que todo está correctamente instalado debemos abrir la terminal y ejecutar estos comandos:

```
$ node -v
$ npm -v
```

Dichos comandos deben de mostrar las versiones de cada recurso.
En caso de que no estén bien instaladas, saldrá un error.

Siempre puedes apoyarte en el profesor en caso de que surja algún problema.

Jest - Framework Testing

Creamos un proyecto en VSC

Inicializamos el proyecto

```
$ npm init
```

Instalamos los siguientes módulos, con el comando:

```
$ npm install --save-dev @types/jest @types/node jest ts-jest typescript
```

Jest - Framework Testing

Para poder ejecutar 'Jest' se necesita editar el fichero y añadir la dependencia.

package.json

En la terminal, abrimos nuestro editor y copiamos el siguiente código:

```
{
  "scripts": {
    "test": "jest"
  }
}
```

Jest - Framework Testing

Creamos el fichero **jest.config.js** para indicar las configuraciones a jest

```
module.exports = {
  "roots": [
    "<rootDir>/src"
  ],
  "testMatch": [
    "**/__tests__/**/*.+(ts|tsx|js)",
    "**/?(*.)+(spec|test).+(ts|tsx|js)"
  ],
  "transform": {
    "^.+\\.(ts|tsx)$": "ts-jest"
  },
}
```

Jest - Framework Testing

Creamos una carpetas **src** y dentro otra **test** .

Dentro de **test** creamos un fichero **sum.test.ts**

```
import { sum } from '../ sum';

test('basic', () => {
  expect(sum()).toBe(0);
});

test('basic again', () => {
  expect(sum(1, 2)).toBe(3);
});
```

Jest - Framework Testing

Dentro de **src** creamos un fichero **sum.ts**

```
export const sum
= (...a: number[]) =>
  a.reduce((acc, val) => acc + val, 0);
```

Ya podemos ejecutar nuestro código con

```
npx jest
```

KATA

Segunda Kata FizzBuzz

Escribe un programa que imprima los números del 1 al 100, pero aplicando las siguientes normas:

- Si no se le pasa nada devuelve 0
- Si se le pasa un 1 devuelve 1
- Devuelve Fizz si el número es divisible por 3.
- Devuelve Buzz si el número es divisible por 5.
- Devuelve FizzBuzz si el número es divisible por 3 y por 5.
- Salida de ejemplo:
    - 1,2,Fizz,4,Buzz,Fizz,7,8,Fizz.

Primera Kata String Calculator

TDD Kata 1 - String Calculator

Create a simple String calculator with a method signature:

---

int Add(string numbers)

- For an empty string it will return 0
- The method can take up to two numbers, separated by commas, and will return their sum.
  - for example “” or “1” or “1,2” as inputs.
- Allow the Add method to handle an unknown amount of numbers

RETO

RETO: Números Primos

Números primos
Un número primo es un número natural que sólo es divisible por 1 y por sí mismo.

Los números que tienen más de un divisor se llaman números compuestos. El número 1 no es ni primo ni compuesto.

Escriba un programa que reciba como entrada un número natural, e indique si es primo o compuesto:

Ingrese un número: `17`
17 es primo

Ingrese un número: `221`
221 es compuesto

RETO kata lonja

En la lonja de un puerto pesquero gallego, un gallego emprendedor ha decidido comercializar pescado fresco gallego en otras ciudades de Europa, para ello va a comenzar con una pequeña furgoneta que es capaz de transportar hasta 200 Kg de pescado.

Para el primer viaje ha comprado 50 Kg de vieiras, 100 Kg de pulpo y otros 50 Kg de centollos, pero se pregunta dónde debería vender esta carga para obtener el máximo beneficio.

RETO kata lonja

Para ello conoce los precios de venta en los diferentes mercados locales:

€/Kg	MADRID	BARCELONA	LISBOA
Vieiras	500	450	600
Pulpo	0	120	100
Centollos	450	0	500

Pero además hay que tener en cuenta que tan sólo cargar la furgoneta le cuesta 5 € más 2 € por cada Km recorrido.

Las distancias hasta los posibles destinos son:

- a MADRID : 800 Km
- a BARCELONA : 1100 Km
- a LISBOA : 600 Km

RETO kata lonja

Y por último, hay que tener en cuenta que los compradores estiman que el valor de compra de la carga se deprecia en un 1% por cada 100 Km recorridos.

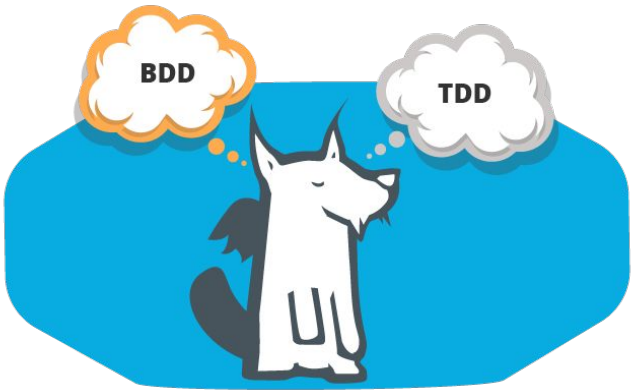
Recordad, el objetivo es responder a nuestro amigo emprendedor dónde debería vender esa carga de pescado y marisco para obtener el mayor beneficio.

BDD

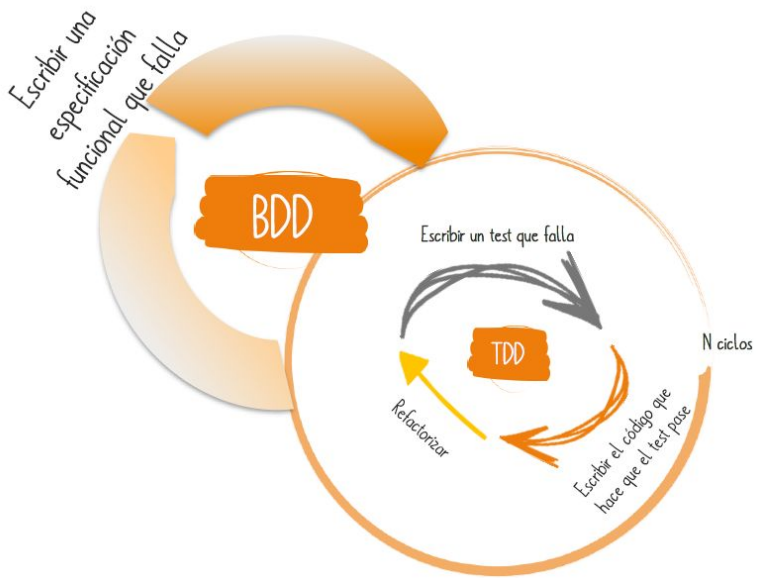
Que es BDD

Behaviour Driven Development (BDD), o desarrollo orientado a comportamiento, es un proceso de desarrollo similar a TDD.

Con BDD lo que se realiza es definir el comportamiento y las especificaciones, al contrario que con TDD que se limita solamente a comprobar funcionalidades.



Que es BDD



¿Cómo se complementa TDD y BDD ?

1. Escribir especificaciones
2. Escribir un test que falla
3. Escribir código que el test pase
4. Refactoring
5. Escribir rectificar especificaciones

Que es BDD

Este sería un ejemplo de cómo se definiría un test BDD

Característica: Crear una calculadora

Feature: Sumar dos números
Instrucción:
Given qué quiero sumar dos números
When introducir el número 2
Them recibo el número 5

KATA

Kata BDD

Inicializamos el proyecto

```
$ npm init
```

Instalamos los siguientes módulos, con el comando:

```
$ npm install --save-dev @types/jest @types/node jest ts-jest typescript

npm i -D jest jest-cucumber typescript ts-jest
```

Kata BDD

Creamos el siguiente árbol de ficheros

- src
  - features
    - step_deinitions
      - sum_a_pair.steps.test.ts
    - sum_a_pair.feature
  - uitls
    - sum.tx

Kata BDD

sum_a_pair.steps.test.ts

```
import { defineFeature, loadFeature } from 'jest-cucumber';
import sum from '../../utils/sum';

const feature = loadFeature('./src/features/sum_a_pair.feature');

defineFeature(feature, test => {
  test('adds 1 + 2 to equal 3', ({ given, when, then }) => {
    let x: number;
    let z: number;
    given('1', () => {x = 1;});
    when('add 2', () => {z = sum(x, 2);});
    then('the sum is 3', () => {expect(z).toBe(3);});
  });
});
```

Kata BDD

sum_a_pair.feature

```
Feature: Sum a Pair
  It sums a pair of numbers

  Scenario: adds 1 + 2 to equal 3
    Given 1
    When add 2
    Then the sum is 3
```

sum.ts

```
export default (a: number, b: number) => a + b;
```

RETO

Kata BDD

Utiliza la kata String Calculator para adaptarla a BDD.

Crea estas dos features de BDD

- Si pasamos vacío devuelve cero
- Si le pasamos un 1,2 devuelve tres

INVESTIGACIÓN

Investigación

Extensiones de VSCode

Instala e investiga estas extensiones de VSC que te pueden ayudar a la hora de realizar tus test

- Jest
- Cucumber (Gherkin) Full Support

https://code.visualstudio.com/Docs/languages/typescript#_refactoring

BIBLIOGRAFÍA

BIBLIOGRAFÍA

- Diseño ágil con TDD
- What is Extreme Programming (XP)?
- When should I use Extreme Programming
- Manifiesto por el Desarrollo Ágil de Software
- BDD + TDD para descubrir el diseño de tu código
- Principios del Manifiesto Ágil
- Scrum y eXtreme Programming para Programadores
- KISS principle
- https://code.visualstudio.com/Docs/languages/typescript#_refactoring