TDD en Javascript

Agenda

1. TDD

- ¿Qué es?
- ¿Cuál es su ciclo?
- ¿Por qué no quiero hacer TDD?
- ¿Por qué quiero hacer TDD?

2. Unit Testing

- ¿Qué es?
- TDD y Unit Testing
- FIRST
- Elementos en el test
- Estructura de un test
- Tipos de colaboradores
- Ejemplos de uso
- Estilos de tests
- Buenas prácticas

Agenda

3. Javascript

- Frameworks de testing
- Tests runners
- Browser Automation
- Assertion Libraries
- Reporters
- Tests automation

4. Demo

- Calculadora
- Calculadora 2

5. Notas finales

- ¿Quiero hacer unit Testing?
- Agradecimientos
- Bibliografía

1. TDD

¿Qué es TDD?

¿Qué es TDD?

Test Driven Development

Es un proceso de desarrollo → programación extrema

¿Qué es extreme programming?

actividades → escuchar al cliente, diseñar, testear y programar

valores → comunicación, simplicidad, feedback, valentía y respeto

principios → feedback, asumir la simplicidad y adoptar el cambio

¿Qué es extreme programming?

objetivo → software de mayor calidad y de forma más productiva

Reducción de coste → + ciclos de cambio cortos que absorban el coste de los cambios en los requisitos del cliente

Elementos → pair programming, code reviews y unit testing

¿Cuál es su ciclo?

¿Cual es su ciclo?

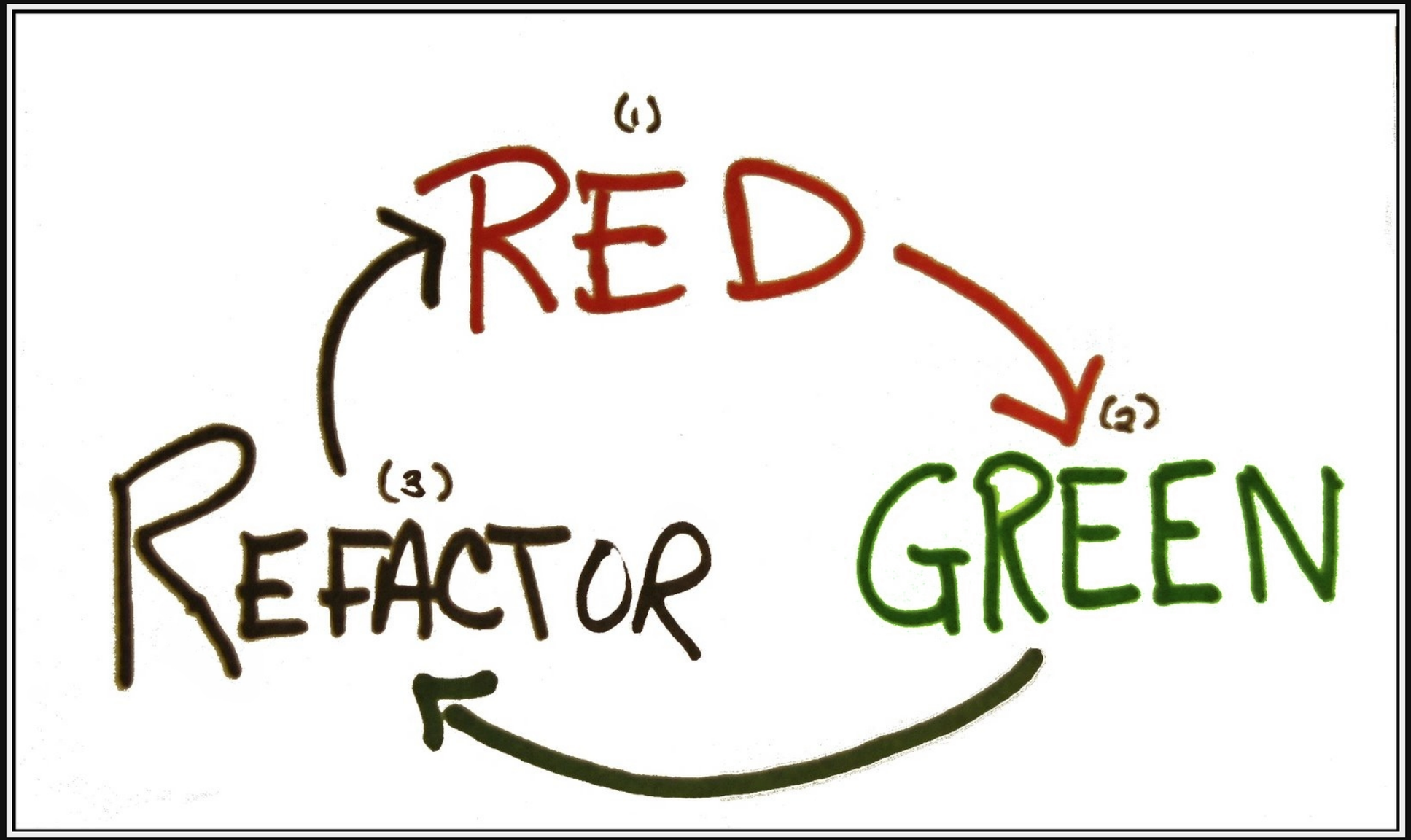
ciclo de desarrollo corto → 3 partes

Implementar el test

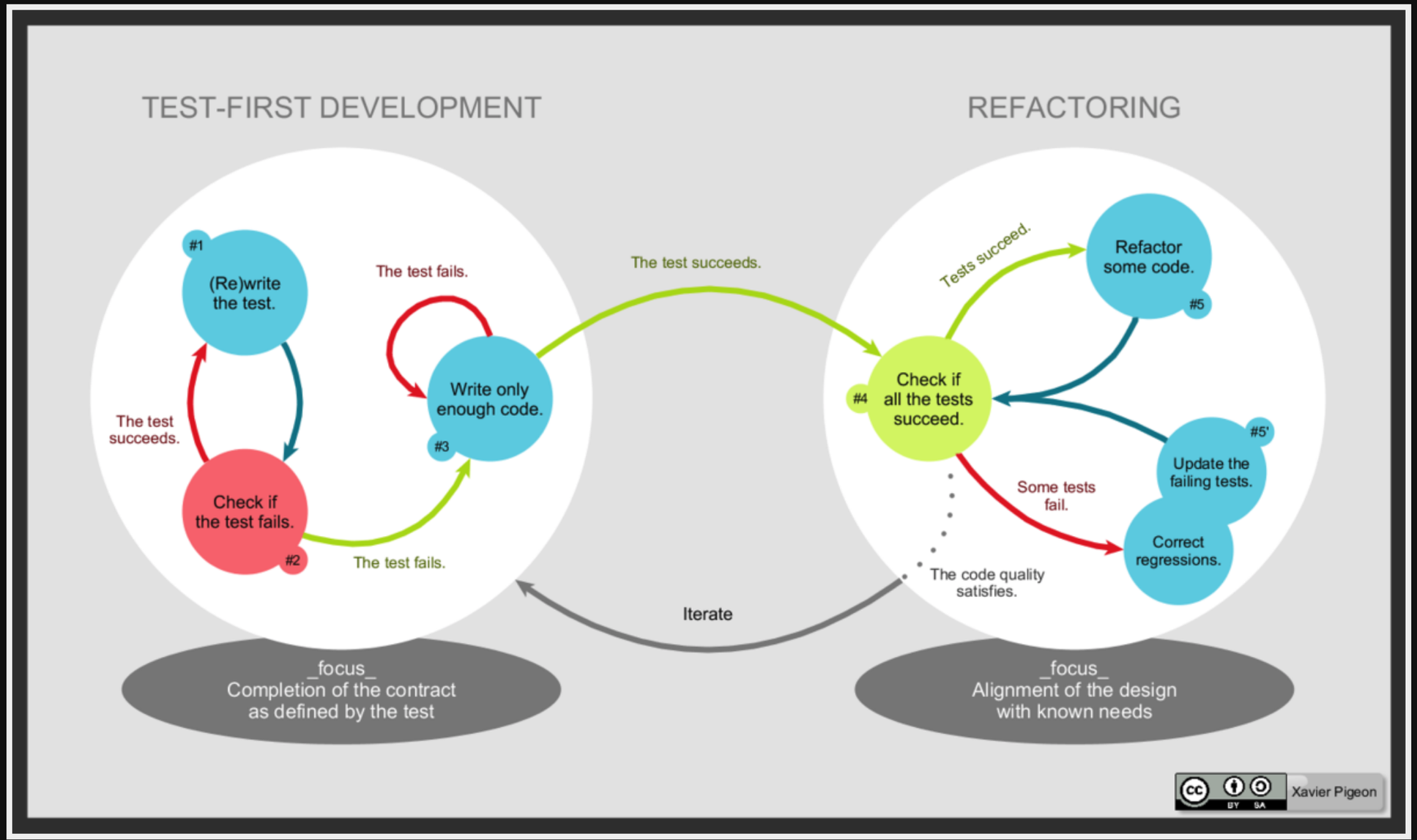
Implementar la funcionalidad

Refactorizar

¿Cual es su ciclo?



¿Cual es su ciclo?



¿Por qué no quiero hacer TDD?

¿Por qué no quiero hacer TDD?

El código y yo

El código y los demás

¿Por qué quiero no hacer TDD?

El código y yo

Perfect is the enemy of good

¿Por qué no quiero hacer TDD?

El código y yo

Puntos ciegos e interpretaciones falsas → puntos ciegos, enfoque de diseño erróneo y de implementación de la solución

¿Por qué no quiero hacer TDD?

El código y los demás

Best costs are no costs

¿Por qué no quiero hacer TDD?

El código y los demás

Falibilidad → tests de usabilidad, accesibilidad, seguridad, de carga...

Coste → tiempo en crearlos y en mantenerlos

Cobertura sesgada → cambios de requisitos constantes

¿Por qué quiero hacer TDD?

¿Por qué quiero hacer TDD?

El código y yo

El código y los demás

¿Por qué quiero hacer TDD?

El código y yo

To create a tiny universe where the software exists to do one thing and do it well

¿Por qué quiero hacer TDD?

El código y yo

Diseño mejorado → resolver el problema antes de implementarlo, casos límite y dependencias

Una tarea a la vez → KISS, YAGNI

Documentación del comportamiento → código + fácil de mantener

Invita al refactor → regresiones en calidad de código fácilmente detectables

Código modularizado, flexible y extensible → integración + sencilla, interfaces + límpias y - acoplamientos

¿Por qué quiero hacer TDD?

El código y los demás

Your manager told you to

¿Por qué quiero hacer TDD?

El código y los demás

Detección temprana de bugs → múltiples módulos interdepedenientes, + regresión, + detectable

Facilita CI → + corrección temprana, - coste por detección conforme avanza en el tiempo

Mayor grado de confianza → batería de tests + refactor === + certeza de detectar errores

Aprobación por parte del cliente → requisito impuesto

2. Unit Testing

¿Qué es Unit Testing?

¿Qué es Unit Testing?

Método de testing de software → tests unitarios

Funcionalidad aislada → clase, módulo, funciones, *it doesn't matter*

1ª Regla del diseño simple → passes all the tests, reveal all intentions, no duplications, fewest elements

TDD y Unit Testing

TDD y Unit Testing

¿Son lo mismo? → NO

TDD se refiere al *cuando*; Unit testing se refiere al *qué*

TDD se centra en el *diseño*; Unit testing se centra en la *funcionalidad aislada*

TDD utiliza tests unitarios, funcionales y de aceptación; Unit testing utiliza tests unitarios

FIRST

FIRST

Criterios de aceptación → **F**ast **I**ndependent **R**epeteable **S**elf-checking **T**imely

Fast → ejecución rápida de forma contínua

Independent → ejecutables en cualquier orden

Repeteable → N veces obtenemos mismo resultado

Self-checking → autoevaluables, pass or fail

Timely → creados de forma paralela al código (¡y en TDD antes!)

Elementos en el test

Elementos en el test

Protagonistas → SUT y Colaboradores

SUT (System Under Test) → sujeto para la realización del test

Colaboradores → objetos que permiten ejercitar el SUT y/o verificar el estado testeado del SUT y el comportamiento de los colaboradores

Elementos en el test

Ejemplo

"Tenemos vuelos, aviones y pasajeros. Queremos probar que, en un vuelo a China, donde la capacidad del avión es de 100 personas, se respeta la política definida para el número de personas permitidas por vuelo"

Elementos en el test

Ejemplo

¿Quién es el SUT? → la política de número de personas del vuelo a China

¿Quién es el colaborador? → el avión que vuela a China

¿Que necesitamos para verificarlo? → un avión con capacidad y estado (lleno/vacío)

¿Cómo lo verificamos? → comprobando el estado de la capacidad

Estructura de un test

Estructura de un test

4 fases → Configuración, ejercitado, Verificado y Reiniciado

Configuración (Setup) → instanciado de los objetos SUT y colaboradores

Ejercitado (Exercise) → SUT y colaboradores realizan la acción a verificar

Verificado (Verify) → comprobación del estado del SUT, colaboradores y/o comportamiento de éstos

Reiniciado (Teardown) → restaura estado del SUT y de los colaboradores

Tipos de colaboradores

Tipos de colaboradores

2 tipos → objetos reales y dobles de tests

Reales → ejercitado y verificado con métodos y estados *propios del objeto*

Dobles de test (Stunt Doubles) → implementan parcialmente métodos, estados o comportamientos *imitando* a los métodos y estados del objeto real

Dobles de tests

Dónde se utilizan

Para qué se utilizan

Dobles de tests

Dónde se utilizan

Dummy → fase de ejercitado, normalmente como parámetro del método

Fake → cualquier fase, no válidos para utilizarlos en producción

Dobles de tests

Para qué se utilizan

Mocks → imitan métodos del objeto real y define expectaciones (comportamiento a priori). Verifican comportamiento

Stubs → imita/reemplaza métodos del objeto real. Verifican estado y comportamiento (a posteriori)

Spies → vigila el comportamiento de los métodos del objeto real. Verifican estado

Ejemplos de uso

Ejemplos de uso

Ejemplo → Objetos reales

```
/**
 * Framework: jasmine
 * SUT: métodos del prototipo area
 * Colaboradores: area
 */

//Fase 1
var area = new Area();
//Suite de tests
describe("Métodos del prototipo Area:\n", function() {
  //Descripción del test
  it("Método areaCuadrado:", function() {
    //Fase 2 y 3
    expect(area.areaCuadrado(2)).toBe(4);
  });
  it("Método areaCirculo:", function() {
    expect(area.areaCirculo(2)).toBe(12.56);
  });
});
```

Ejemplos de uso

Ejemplo → Dummy

```
/**
 * Framework: jasmine
 * SUT: métodos del prototipo volumen
 * Colaboradores: areaStub y volumen
 */

//Fase 1
var areaStub = {
  areaCuadrado: function(lado) {
    //lado = 2
    return 4;
  },

  areaCirculo: function(radio) {
    //radio = 3
    return 28.26;
  }
};

var volumen = new Volumen();
```

Ejemplos de uso

Ejemplo → Spies

```
/**
 * Framework: Sinon
 * SUT: métodos del módulo area
 * Colaboradores: area y spy
 */

var area = require('../src/js/area.js');

describe('Utilizando spies para areaCuadrado()', function() {
  it('Se llama una vez, con parámetro 2 y debe ser igual a 4',
    function() {
      //given
      var spy = sinon.spy(area, 'areaCuadrado');
      //when
      area.areaCuadrado(2);
      //then
      assert(spy.callCount === 1);
      assert(spy.calledOn(area));
      assert(spy.calledWith(2));
      assert(spy.withArgs(2).calledOnce);
```

Ejemplos de uso

Ejemplo → Stubs

```
/**
 * Framework: Sinon
 * SUT: métodos del módulo volumen
 * Colaboradores: pruebaArea, pruebaVolumen y stub
 */

var pruebaArea = require("../src/js/area.js");
var pruebaVolumen = require("../src/js/volumen.js");

describe('Stubbing areaCirculo', function() {
  it('Stub con \'returns\'', function() {
    var stub = sinon.stub(pruebaArea, 'areaCirculo');
    stub.withArgs(4).returns(10);
    assert.equal(53.2, pruebaVolumen.volumenEsfera(4));
    stub.restore();
  });

  it('Stub con una nueva función', function() {
    var stub = sinon.stub(pruebaArea, 'areaCirculo', function() {
      return 10;
```

Ejemplos de uso

Ejemplo → Mocks

```
/**
 * Framework: Sinon
 * SUT: métodos del módulo volumen
 * Colaboradores: area, volumen y mock
 */

var area = require("../src/js/area.js");
var volumen = require("../src/js/volumen.js");

describe('Mocking areaCuadrado()', function() {
  it('calls areaCuadrado() when calls volumenCubo()', function() {
    var mock = sinon.mock(area);
    var expectation = mock.expects('areaCuadrado');

    expectation.once().withArgs(3);
    expectation.withArgs(3).returns(9);
    volumen.volumenCubo(3);
    mock.verify();
  });
});
```

Estilos de tests

Estilos de tests

2 escuelas → Classic & Mockist

Classic → utilizan **tests sociables**. Prefieren la utilización de objetos reales a utilizar test doubles, salvo que la colaboración de éstos sea complicada

Mockist → utilizan **tests solitarios**. Prefieren la utilización de tests doubles que la utilización de objetos reales

Buenas prácticas

Buenas prácticas

Enfoque → legibilidad y mantenibilidad

DAMP → *D*escriptive *A*nd *M*eaningful *P*hrases

Rule of thumb → 1 afirmación *Ó* 1 verificación por test

Evita código duplicado → utiliza hooks *beforeEach & afterEach*

3 A's → Arrange, Act & Assert

No testees los dobles de test → son colaboradores no son SUT!

3. Javascript

Javascript

Primeras impresiones → Welcome to the Jungle

Nombres descriptivos → Mocha, Jasmine, Karma

1 única responsabilidad → Jest, Ava, Mocha

Conocidos de toda la vida → frameworks, tests runners, browser automation, assertion libraries, automation testing

¡Tó guapo!

Frameworks de testing

Frameworks de testing

Responde → ¿Cómo defino mis tests unitarios?

¿Qué son? → API de definición de tests con *esteroides*

¿Para qué sirven? → definición de tests, soporte de navegadores y/o node, reportes, integración con CI, presentación formateada en CLI (o con CLI integrado), búsqueda y filtrado de errores...

¿Dependientes del framework? → Jest

¿Más conocidos? → Jasmine, Mocha, Qunit, Ava, Jest, Tape

Tests runners

Tests runners

Responde → ¿Cómo lanzo mi batería de tests?

¿Qué son? → herramientas para el lanzamiento de tests y generación de reporters

¿Para qué sirven? → navegadores, dispositivos, durante el desarrollo, en un entorno de CI, sincronizarlo con reporters...

¿Dependientes del framework? → WCT (Polymer)

¿Más conocidos? → Karma, Buster, Intern, Testswarm, Mocha, Ava

Browser automation

Browser automation

Responde → ¿Cómo pruebo mis tests en los navegadores?

¿Qué son? → herramientas para la definición y automatización del lanzamiento de tests frente al navegador

¿Para qué sirven? → unit testing frente a elementos DOM, E2E testing, soporte cross browsing, headless browsing...

¿Dependientes del framework? → Protactor (Angular)

¿Más conocidos? → Selenium (webDriver), Testcafe, NightWatch, PhantomJS, ZombieJS

Assertion Libraries

Assertion Libraries

Responde → ¿Qué puedo hacer dentro de mi test?

¿Qué son? → librerías que ofrecen API's específicas para la definición de tests con un lenguaje más definido

¿Para qué sirven? → definición de colaboradores, distintos tipos de verificaciones más específicos

¿Dependientes del framework? → Enzyme (React)

¿Más conocidos? → Chai, Sinon, Expect

Reporters

Reporters

Responde → ¿Cómo puedo extraer información del resultado de los tests?

¿Qué son? → herramientas para la obtención de reportes de tests

¿Para qué sirven? → principalmente para integrar el desarrollo de tests unitarios con herramientas de CI y calidad de código (cobertura, complejidad)

¿Dependientes del framework? → ¿conocéis alguno?

¿Más conocidos? → jUnit (xml), jasmine-reporters, mocha-reporters, karma-reporters, istanbul-reporters (lcov)

Tests automation

Tests automation

Responde → ¿Cómo integro los tests con mi desarollo?

¿Qué son? → herramientas para automatizar el lanzado de tests

¿Para qué sirven? → para integrar el desarrollo de tests con el desarrollo de código, a distintos niveles (testing, desarollo)

¿Dependientes del framework? → dependen del runner, framework

¿Más conocidos? → gulp, grunt, karma-adapters

4. Demo time

5. Notas finales

¿Quiero hacer Unit Testing?

¿Quiero hacer Unit Testing?

Ventajas

Me da confianza. Sé lo que funciona y lo qué no

Me ayuda a encontrar errores en el código

Me invita a refactorizar, a mejorar el código

Me ayuda a documentar el código y hacerlo más entendible

Me permite añadir nuevas funcionalidades sin miedo

Me permite hacer pingpong pair programming

Me divierto y aprendo haciéndolo

Pero recuerda...

No es infalible

¿Quiero hacer Unit Testing?

Consejos

Testea código que sea complejo y que cambie de forma frecuente

Nunca testees legacy code que no ha sido testeado previamente, ni código de 3ª partes, salvo que sea estrictamente necesario

Nunca testees código que requiera crear el test con un orden de mágnitud más complejo que el propio código

Nunca testees código "throw-away" ó "placeholder"

¿Quiero hacer Unit Testing?

Martin Fowler said...

Tests don't prove the absence of bugs

Imperfect tests, run frequently, are much better than perfect tests that are never written at all

The key is to test the areas that you are most worried about going wrong. That way you get the most benefit for your testing effort

ATTITUDE IS EVERYTHING.

¡Muchas gracias a tod@s!

Bibliografía

Bibliografía

Libros

Extreme Programming Explained: Embrace Change

The pragmatic programmer

Software Craftsman

Learning JavaScript Design Patterns

Javascript Testing Recipes

Test Driven Javascript Development

Testable Javascript

Bibliografía

Artículos I

Extreme programming: a gentle introduction

BeckDesignRules, Unit Test, SelfTestingCode, Mocks Aren't Stubs

The Little Mocker

5 Common Misconceptions About TDD & Unit Tests

The false dichotomy of tests

Bibliografía

Artículos II

One weird trick that will change the way you code forever: Javascript TDD & TDD should be fun

F.I.R.S.T Principles of Unit Testing

JavaScript Testing: Unit vs Functional vs Integration Tests

5 Questions Every Unit Test Must Answer

Writing testable Javascript

The Plight of Pinocchio: JavaScript's quest to become a real language

Testable JavaScript - Architecting Your Application for Testability

Bibliografía

Cursos y tutoriales

Let's code Javascript

Testing Clientside JavaScript

Javascript Testing (Udacity)

Introduction to Test Driven Development

TDD Javascript series(I, II, III)

Writing Testable Frontend Javascript (I, II)

Bibliografía

Tutoriales y repositorios

Charla y slides

Learn Test Driven Development (TDD)

A test-driven JS assessment

Javascript: Test Driven Learning

Awesome TDD en Javascript

js-unit-testing-guide

Javascript Testing Frameworks

Bibliografía

Karma, Jasmine, Mocha y Chai

JavaScript Testing with Mocks, Spies & Stubs

Sinon Spies vs. Stubs

Best Practices for Spies, Stubs and Mocks in Sinon.js

Unit Test Your JavaScript Using Mocha and Chai

AJAX and Sinon.js