Testing con Javascript

KEEPCODING
Tech School

Contenido

- Qué es el testing
- Tipos de testing
- Herramientas para escribir y ejecutar tests
- Resumen

Qué es el testing

Qué es el testing

Es una **buena práctica** de los desarrolladores para dotar de robustez y solidez a las aplicaciones que creamos.

¿Es necesario entonces?

¿Puede un código no tener conjunto de tests?

Es una cuestión **CULTURAL**

Objetivos del testing

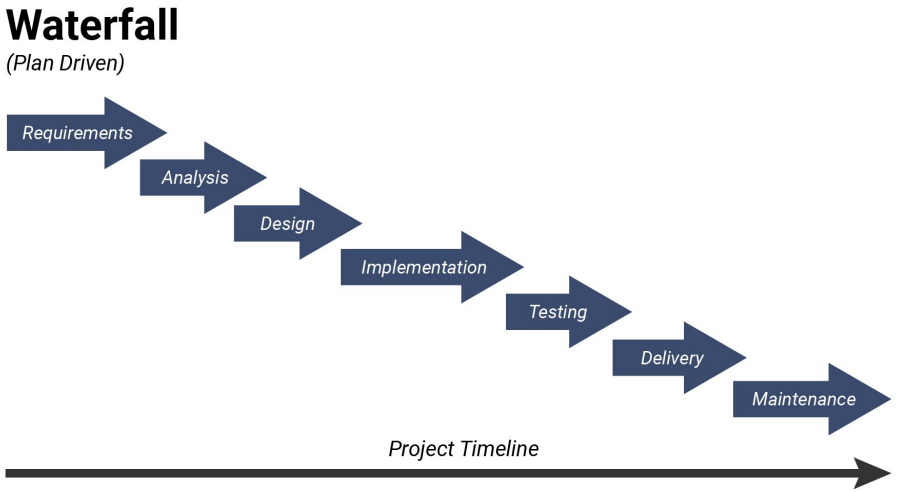
- Encontrar **bugs**
- Nos permite tener más **confianza en el código.** Aunque nunca vamos a poder afirmar al 100% que nuestra app no contiene errores, sabemos que ese código está comprobado en su mayor parte antes de lanzarlo a producción
- Lógicamente, el testing nos ayudará a **decidir si es adecuado subir ciertas funcionalidades a producción** o, por el contrario, esperar más tiempo y subsanar los errores que podamos tener.
- **Evitar la aparición de defectos en la aplicación**. Al hacer testing te acostumbras a escribir código de una manera más limpia, ordenada y con una mayor calidad.

Relación con el PM

- **El testing** es una pieza relevante en cualquier tipología de gestión de proyecto. En la cadena de producción, sea cual sea, siempre se puede/debe introducir la fase de testing.
- Ejemplos de la relación entre el testing y tipos básicos de gestión de proyectos
  - Waterfall
  - Agile
  - DevOps

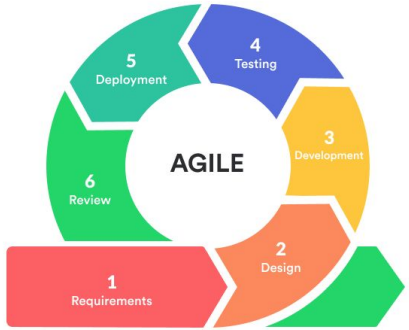
Relación con Waterfall

- En proyectos con esquema Waterfall el testing es una pieza más de la cadena de producción. Puede ,o no, estar presente.



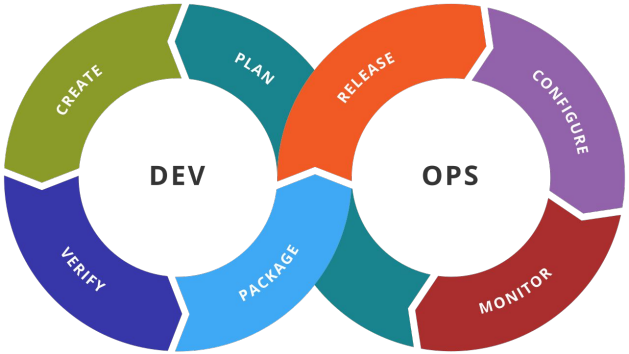
Relación con Agile

- Agile, al ser un framework con métodos iterativos, permite introducir el testing como herramienta fundamental en los ciclos de producción.
- Conseguimos desarrollar de forma más eficiente. Al mejorar (proceso incremental), si se rompe algo, lo vemos al momento.
- Tiene una estrecha relación con el TDD.



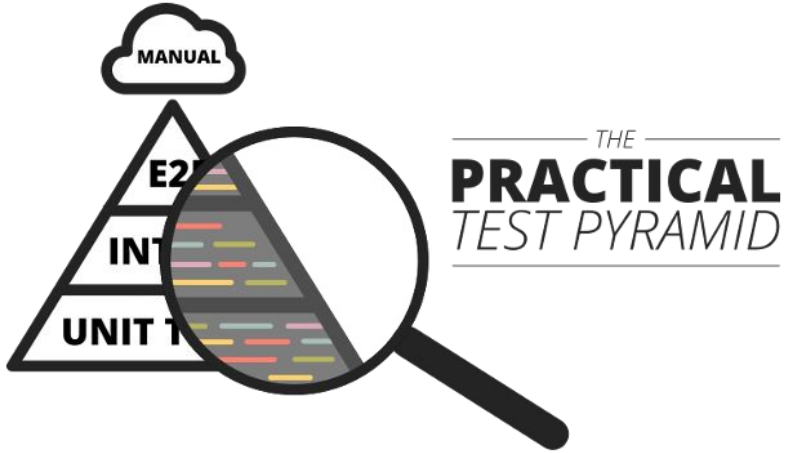
Relación con DevOps

- Es un elemento imprescindible en el loop infinito de producción.
- Como el valor añadido de DevOps pasa por la automatización de procesos, es obligatorio contar con elementos de testing.



Tipos de testing

Tipos de testing



https://martinfowler.com/articles/practical-test-pyramid.html

Tests externos

- Son test realizados por herramientas externas.
- Estas herramientas no puede acceder al código de nuestra aplicación, por lo que son tests que comprueban la experiencia de usuario y el correcto funcionamiento de nuestra web desde una parte visual. Se dividen en dos tipos:
  - Manuales: Se siguen una serie de pasos y se comprueba la funcionalidad
  - Automáticas: Grabamos una interacción con nuestra app y podemos especificar cada cuanto tiempo queremos que se vuelva a realizar esta comprobación
- Ejemplo: https://www.pingdom.com/

Tests funcionales

- A diferencia de los tests externos, estos se basan en la parte funcional de nuestra aplicación, por lo tanto, testeamos nuestro código.
- Tipos de tests funcionales:
  - Unitarios
  - Integración
  - Aceptación
  - Regresión
  - e2e

Tests unitarios

- Son los tests más usados habitualmente (jest, mocha, jasmine…).
- Consiste en comprobar cada pequeña porción de código de nuestra app por separado para asegurarnos de su correcto funcionamiento.
- **Cometido único**: Estos test no deben comprobar varias cosas a la vez, cada test debe centrarse en algo muy específico y debemos escribir un test por cada funcionalidad que queramos testear.
- **Múltiples casos**: Puede que esa pequeña parte de código que estemos testeando, funcione diferente dependiendo de los parámetros que reciba, por ello debemos comprobar de manera individual cada uno de esos posibles casos para certificar el correcto funcionamiento de esa parte del código.

Tests de integración

- Son pruebas en las que se comprueba el funcionamiento entero de nuestra aplicación.
- Con unit testing testeamos cada una de las piezas. Ahora nos centramos en unir **dos o más** de esas piezas y comprobar que funcionan todas en conjunto.
- Es muy frecuente que aunque algo funcione individualmente de la forma esperada, a la hora de integrarlo con el resto de nuestra app esa función deje de funcionar por una razón externa a sí misma.
- En el caso del back-end, consistiría en realizar peticiones http a nuestra api y comprobar que responde de manera correcta con todos los parámetros que debería devolver.

Tests de aceptación

- Sirven para verificar que un sistema cumple con los requerimientos de negocio definidos (ejemplo: un listado tarda demasiado en cargar).
- Suelen realizarse luego de las pruebas unitarias o de integración, para evitar que se avance mucho con el proceso de prueba, y determinar a tiempo si se necesitan cambios significativos.
- Los tests de aceptación suelen ser definidos y ejecutados por usuarios externos.

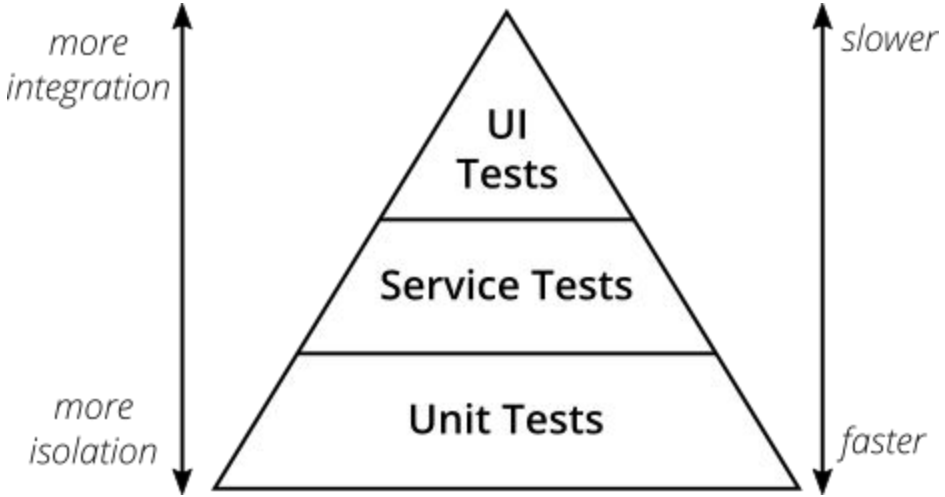
Tests de regresión

- Los tests de regresión se realizan cuando hemos introducidos cambios en nuestra app y, consisten en comprobar que partes de nuestra app, que no han sido tocadas en esos cambios que hemos introducido, siguen funcionando correctamente después de haber introducido esas modificaciones.
- Sirven para verificar que el código sigue funcionando para versiones antiguas de software.

Tests E2E

- Son tests que replican comportamientos de usuario.
- Se conocen también como pruebas de alto nivel.
- Tienen por objetivo verificar que el sistema responde de forma correcta a los comportamientos del usuario (ej: login de un usuario, añadir items al carrito de la compra, pasarelas de pago, etc).
- Son costosas de producir y de mantener. Se recomienda tener pocas en general.

Resumen test funcionales



https://martinfowler.com/articles/practical-test-pyramid.html

Tests no funcionales

- Son tests que no tienen por objetivo analizar la funcionalidad del código.
- Tipos de tests funcionales:
  - Seguridad
  - Rendimiento
  - Usabilidad
  - Accesibilidad

Tests de seguridad

- El objetivo es comprobar las posibles vulnerabilidades que pueda tener nuestra app.
  - Paquetes y dependencias.
  - Encriptación.
  - Datos sensibles.

Tests de rendimiento

- El objetivo es comprobar la performance de nuestro código.
  - TTF (time to first byte).
  - Cálculo de costes de operación.

Tests de usabilidad

- Conocer si determinadas acciones realizadas por usuarios reales son intuitivas o no, si hay impedimentos, etc.

Tests de accesibilidad

- Se testea si se renderizan bien todos los elementos de una forma visual, además de comprobar si la aplicación tiene alternativas en sus funcionalidades para que puedan ser usadas por personas con algún tipo de discapacidad.

Herramientas

Intro

Podemos distinguir dos tipos de herramientas: imprescindibles, frameworks de testing (o herramientas técnicas).

- Imprescindibles:
  - Conocer el lenguaje de programación que se va a testear.
  - Conocer el código que se va a testear.
- Software:
  - Tests unitarios
  - Testing funcional
  - Tests e2e

Software

- Unit testing:
  - Jasmine
  - **Jest**
  - Mocha
  - Chai
- Integración
  - **Supertest**
- E2E
  - Selenium
  - **Puppeteer**
  - Protractor (Angular)

Unit testing: Jasmine

Jasmine es framework que se basa en el comportamiento del código JavaScript dentro de una app.

- No depende de ningún otro marco de JavaScript
- No requiere un DOM
- Tiene una sintaxis clara y obvia para que puedas escribir fácilmente las pruebas.
- Según sus creadores, Jasmine es:
  - Rápido: no requiere de dependencias externas.
  - Completo: contiene todo lo necesario para la realización de los tests.
  - Multiusos: sirve tanto para front como para back.

Unit testing: Jasmine

```
describe("A suite is just a function", function() {
  var a;

  it("and so is a spec", function() {
    a = true;

    expect(a).toBe(true);
  });
});
```


Unit testing: Jest

- Desarrollado por facebook (igual que React).
- En un primer lugar, nació como framework de testing para el entorno de React.
- Pronto se convirtió en un framework de testing generalista.
- En los último años, se ha establecido como un standard para el testing en la comunidad de Javascript.
- Su primera versión fue lanzada el 14 de mayo de 2014.
- Destaca por:
  - Sintaxis clara
  - Contextos únicos
  - Tests descriptivos

Unit testing: Jest

```
describe("Creating a new group", async function() {
  const group = await makeAuthenticatedQuery(createGroup, {
    iconUrl: "",
    name: "",
    description: ""
  });

  test("should exist", function() {
    expect(group.id).toBeUndefined();
  });
});

```


Unit testing: Mocha

- Se describe como un framework de testing tanto para NodeJS como para el navegador.
- Es un duro competente de Jest.
- Su primera versión se lanzó el 22 de noviembre de 2011.
- Ejecuta los tests en serie, es decir, hasta que no finaliza uno no pasa al siguiente facilitando que la detección de fallos en nuestros tests sea más precisa.
- En principio, su punto fuerte fue la posibilidad de ejecutar acciones asíncronas en los tests y poder comprobar su resultado.
- Permite usar cualquier dependencia para la comprobación de los test, por ejemplo assert.

Unit testing: Mocha

```
var assert = require('assert');
describe('Array', function() {
  describe('#indexOf()', function() {
    it('should return -1 when the value is not present', function() {
      assert.equal([1, 2, 3].indexOf(4), -1);
    });
  });
});
```

Unit testing: Chai

- También se describe como un framework de testing tanto para NodeJS como para el navegador.
- Es muy parecido a mocha.
- Expande su librería de testing a should, expect (orientado a BDD) y también provee sintaxis assert.

Unit testing: Chai

Should

```
chai.should();

foo.should.be.a('string');
foo.should.equal('bar');
foo.should.have.lengthOf(3);
tea.should.have.property('flavors')
  .with.lengthOf(3);
```

Visit Should Guide

Expect

```
var expect = chai.expect;

expect(foo).to.be.a('string');
expect(foo).to.equal('bar');
expect(foo).to.have.lengthOf(3);
expect(tea).to.have.property('flavors')
  .with.lengthOf(3);
```

Visit Expect Guide

Assert

```
var assert = chai.assert;

assert.typeOf(foo, 'string');
assert.equal(foo, 'bar');
assert.lengthOf(foo, 3)
assert.property(tea, 'flavors');
assert.lengthOf(tea.flavors, 3);
```

Visit Assert Guide

Integration testing: Supertest

- La versión 0.1 fue lanzada el 2 de julio de 2012 pero no liberaron la versión 1.0 hasta el 11 de mayo de 2015.
- Se define como un framework que permite una abstracción del API usando peticiones HTTP.
- Se usa junto con Mocha o Jest para comprobar el resultado exacto de las peticiones.
- Se una para realizar test de integración.
- Hoy en día, no debería haber ninguna app sin implementar test con supertest porque gracias a él puedes comprobar lo que devuelve tu API en todo momento y asegurarte de que no va a tener ningún comportamiento inesperado.

Integration testing: Supertest

```
const request = require('supertest')
const app = require('../server')
describe('Post Endpoints', () => {
  it('should create a new post', async () => {
    const res = await request(app)
      .post('/api/posts')
      .send({
        userId: 1,
        title: 'test is cool',
      })
    expect(res.statusCode).toEqual(201)
    expect(res.body).toHaveProperty('post')
  })
})
```

Testing E2E: Selenium

- Es una herramienta de automatización de tests generando browsers reales.
- Muy útil para tests de regresión.
- Dispone de un IDE.

```
const By = webdriver.By; // useful Locator utility to describe a query fo
// open a page, find autocomplete input by CSS selector, then get its valu
driver
  .navigate()
  .to("http://path.to.test.app/")
  .then(() => driver.findElement(By.css(".autocomplete")))
  .then(element => element.getAttribute("value"))
  .then(value => console.log(value));
```

Testing E2E: Puppeteer

- Es una herramienta de automatización de tests generando respaldado en Chrome.
- Lo que conseguimos es usar el developer tools de chrome.

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

(async () => {
  const browser = await puppeteer.launch();
  const page = await browser.newPage();
  await page.goto('https://example.com');
  await page.screenshot({path: 'example.png'});

  await browser.close();
})();
```

Testing E2E: Protractor

- Es una herramienta para testear AngularJS y Angular.
- Utiliza un browser real.
- Depende de selenium.

```
describe('angularjs homepage todo list', function() {
  it('should add a todo', function() {
    browser.get('https://angularjs.org');

    element(by.model('todoList.todoText')).sendKeys('write first protractor test');
    element(by.css('[value="add"]')).click();

    var todoList = element.all(by.repeater('todo in todoList.todos'));
    expect(todoList.count()).toEqual(3);
    expect(todoList.get(2).getText()).toEqual('write first protractor test');

    // You wrote your first test, cross it off the list
    todoList.get(2).element(by.css('input')).click();
    var completedAmount = element.all(by.css('.done-true'));
    expect(completedAmount.count()).toEqual(2);
  });
});
```

Resumen

Resumen

- El uso de testing está cada vez más integrado en las cadenas de producción y en los diferentes modelos de gestión de proyectos.
- Existen diferentes tipos de testing. Cada tipo de testing tiene su propio cometido.
- Para cada tipo de testing podemos contar con diferentes herramientas.
- Podemos testear tanto elementos de backend como de frontend.

Resumen

- ¿Se debe testear todo?
- ¿Es viable dedicar recursos al testing?

Beneficios del testing:

- Robustez
- Agilidad
- Mejora la documentación
- Se destila la delegación de responsabilidades

Contras del testing:

- Necesita recursos (tiempo, organización… en definitiva, €)

DESCANSO

KEEPCODING

Tech School

Madrid | Barcelona | Bogotá

Datos de contacto