Assembler
School of Software Engineering

Unit Test And TDD

Who Am I

- ➢ Full Stack Developer
- ➢ DB Training - Deutsche Bahn (Freelance)
- ➢ Functional programming enthusiast.
- ➢ Believer of the try until you mastered
- ➢ Curious by nature
- ➢ Addicted to coffee

A brief introduction

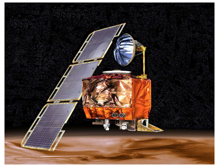
Software testing is an activity to check whether the actual results match the expected results and to ensure that the software system has no issues

July 2019, users across the globe found themselves unable to load photos in the Facebook News Feed, view stories on Instagram, or send messages in WhatsApp. Although Facebook didn't detail the reason for the outage, it did release a statement claiming the issue had been accidentally triggered during "routine maintenance".

HSBC

2016, HSBC became the first bank to suffer a major IT outage. Millions of the bank's customers were unable to access online accounts. Services only returned to normal after a two-day outage.

December 2015 a glitch caused more than 3,200 US prisoners to be released early. The software calculates a prisoner's sentence depending on good/bad behaviour.



1998 Climate Orbiter spacecraft on its mission to Mars was ultimately lost in space. An engineering team failed to make a simple conversion from English units to metric. An embarrassing lapse that sent the $125 million craft fatally close to Mars' surface after attempting to stabilize its orbit too low.

Types of Software Testing

Functional testing:

Testing all the functionalities by providing appropriate input to verify whether the actual output is matching the expected output or not, each function of the software application behaves as required

Non-functional testing:

Refers to various aspects of the software such as performance, load, stress, scalability, security, compatibility etc., Main focus is to improve the user experience on how fast the system responds to a request.

- **Unit Testing:** Unit Testing is done to check whether the individual modules of the source code are working properly.

- **Integration Testing:** Integration Testing is the process of testing the connectivity or data transfer between a couple of unit tested modules.

- **System Testing (end to end testing):** Testing the fully integrated application to ensure that the software works in all intended ways. Verify every input in the application to check for desired outputs. Testing of the users experiences with the application.

- **Acceptance Testing:** To obtain customer sign-off so that software can be delivered and payments received.

Unit Test

It is an automated piece of code that invokes a unit in the system and then checks a single expectation about the behavior of that unit.

Benefits of Unit Testing

- Increases confidence in changing or maintaining code.
- Debugging is easy. When a test fails, only the latest changes need to be checked.
- The effort required to find and fix defects found during unit testing is very less in comparison to the effort required to fix defects found during system testing or acceptance testing.
- Makes refactoring easier.
- Can be a good reference for **documentation.**
- New and regression defects can be minimized.
- Codes are more reusable.

How to write a good unit test

- ➢ Fully automated and repeatable.
- ➢ No side effects. Consistently returns the same result (no random results).
- ➢ Full control over all the unit running (Mocks or Stubs when needed).
- ➢ Runs fast.
- ➢ Can be run in any order if part of many other tests.
- ➢ Runs in memory (no DB or File access, for example).
- ➢ Write small pieces (units) of code. Tests a single logical concept in the system.
- ➢ Readable and Maintainable.

Advices

- All public methods should be tested.
- All the suit of test should be green ideally before pushing to master or feature branch.
- Automatic regression detection. If you write tests for every bug you fix, one thing passing your unit tests can guarantee is that you haven't reintroduced an old bug.
- Define a naming convention for the test
- Mock out all service calls and database operations.
- Thinking about domain area and scope of the written code.
- Write down positive cases
- Write down corner cases. Make sure the parameter boundary cases are covered.
  - Numbers,(negatives, 0, positive, smallest, largest, NaN, infinity).
  - Strings (empty string, single character string, non-ASCII string.
  - Collections (empty, one, first, last.
  - Dates (formats).
- Negative cases. Intentionally misuse the code and verify robustness and appropriate error handling.
- Write only one desired output or result.

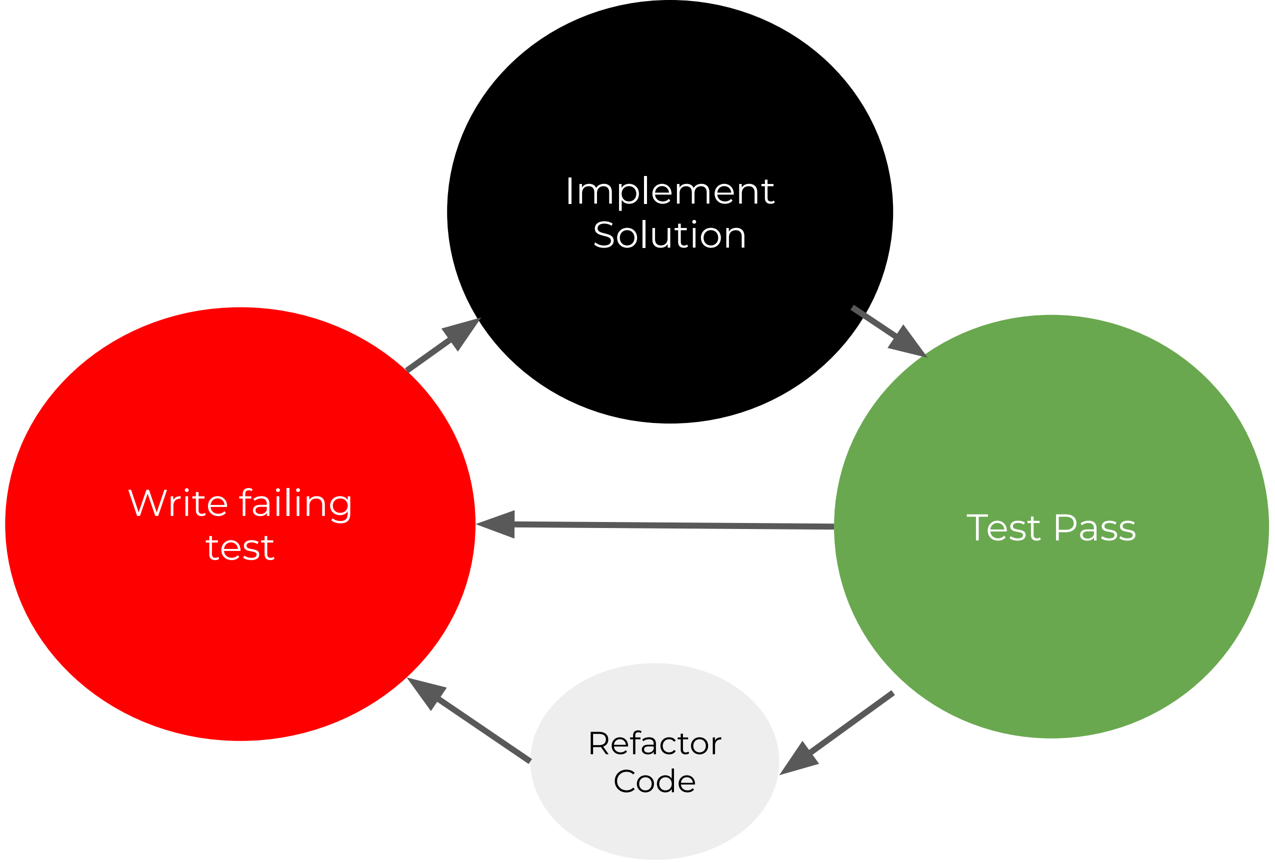
Frameworks

- **MochaJS** is a JavaScript test framework running on Node.js and in the browser, making asynchronous testing simple.
- **Jasmine** is a behavior-driven development framework for testing JavaScript code. It does not depend on any other JavaScript frameworks. It does not require a DOM.
- **AVA** is a minimalistic light-weight testing framework that leverages asynchronous nature of Javascript. AVA can perform tests concurrently.
- **JEST** is one of the most popular frameworks that is maintained regularly by Facebook. It is a preferred framework for the React based applications as it requires zero configuration.
- **Karma** is a productive testing environment that supports all the popular test description framework within itself. It has wide support for executing tests on different devices and applications.
- **Cypress** It is an independent test runner that does not need to integrate with your code closely.
- **Puppeteer** Built by a team at Google. It provides a headless chrome API for NodeJS applications. It is primarily used for applications specific to the browser.
- **ChaiJS** framework focuses on providing assertions for other frameworks.

TEST DRIVEN DEVELOPMENT (TDD)

- It’s one of the core practices of Extreme Programming (XP).
- It’s a software development technique that merges design, implementation, and testing in a progression of small iterations that focus on simplicity and feedback.
- You have to think of an outcome you want which you don’t currently have
- Uses a “test first” approach in which test code is written before the actual code.
- They are written one at a time , followed by the code required to get the test case to pass.

TDD gives developers the power to think first how to do things before they engage and compromise themselves with the implementation.


Implement Solution
Test Pass
Write failing test
Refactor Code

Jest

- ➢ Jest is an amazing testing library created by Facebook to help test JavaScript code.
- ➢ Made for React.
- ➢ Familiar syntax.
- ➢ Great reports.
- ➢ Meaningful error messages.
- ➢ Configurable
- ➢ Your tests run in parallel so they are executed much faster than other testing frameworks.

Jest Matchers

```
test('null', () => {
  const n = null;
  expect(n).toBeNull();
  expect(n).toBeDefined();
  expect(n).not.toBeUndefined();
  expect(n).not.toBeTruthy();
  expect(n).toBeFalsy();
});

test('zero', () => {
  const z = 0;
  expect(z).not.toBeNull();
  expect(z).toBeDefined();
  expect(z).not.toBeUndefined();
  expect(z).not.toBeTruthy();
  expect(z).toBeFalsy();
});
```

```
test('two plus two', () => {
  const value = 2 + 2;
  expect(value).toBeGreaterThan(3);

expect(value).toBeGreaterThanOrEqual(3.5
);
  expect(value).toBeLessThan(5);

expect(value).toBeLessThanOrEqual(4.5);

  // toBe and toEqual are equivalent for
numbers
  expect(value).toBe(4);
  expect(value).toEqual(4);
});
```

```
test('there is no I in team', () => {
  expect('team').not.toMatch(/I/);
});

test('but there is a "stop" in
Christoph', () => {
  expect('Christoph').toMatch(/stop/);
});
```

Assembler
School of Software Engineering

< CODE YOUR FUTURE />

Jest Matchers

```
const shoppingList = [
  'diapers',
  'kleenex',
  'trash bags',
  'paper towels',
  'beer',
];

test('the shopping list has beer on it',
() => {

expect(shoppingList).toContain('beer');
  expect(new
Set(shoppingList)).toContain('beer');
});
```

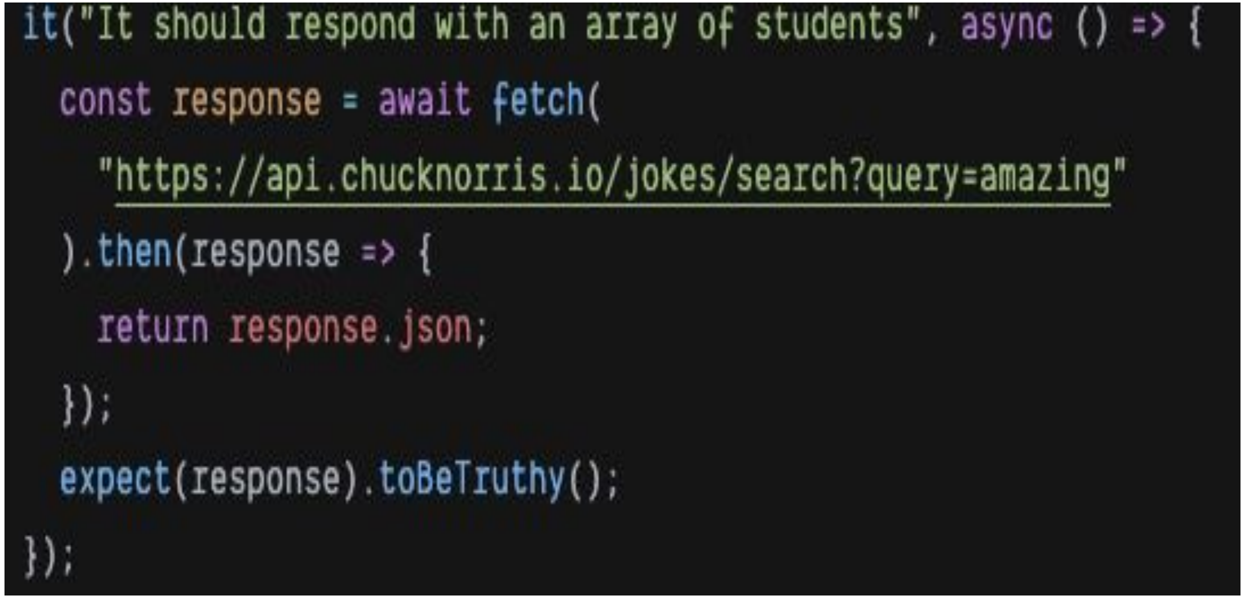
```
function compileAndroidCode() {
  throw new Error('you are using the
wrong JDK');
}

test('compiling android goes as
expected', () => {
  expect(compileAndroidCode).toThrow();

expect(compileAndroidCode).toThrow(Error
);

  // You can also use the exact error
message or a regexp

expect(compileAndroidCode).toThrow('you
are using the wrong JDK');

expect(compileAndroidCode).toThrow(/JDK/
);
});
```

Assembler
School of Software Engineering

< CODE YOUR FUTURE />

Jest Setup

- ➢ beforeAll - called once before all tests.
- ➢ beforeEach - called before each of these tests (before every test function).
- ➢ afterEach - called after each of these tests (after every test function).
- ➢ afterAll - called once after all tests.



```
it("It should respond with an array of students", async () => {
  const response = await fetch(
    "https://api.chucknorris.io/jokes/search?query=amazing"
  ).then(response => {
    return response.json;
  });
  expect(response).toBeTruthy();
});
```

Snapshot Test

Snapshot tests are a very useful tool whenever you want to make sure your UI does not change unexpectedly.

```
import React from 'react';
import Link from '../Link.react';
import renderer from 'react-test-renderer';

it('renders correctly', () => {
  const tree = renderer
    .create(<Link page="http://www.facebook.com">Facebook</Link>)
    .toJSON();
  expect(tree).toMatchSnapshot();
});
```

```
exports[`renders correctly 1`] = `
<a
  className="normal"
  href="http://www.facebook.com"
  onMouseEnter={[Function]}
  onMouseLeave={[Function]}
>
  Facebook
</a>
`;
```

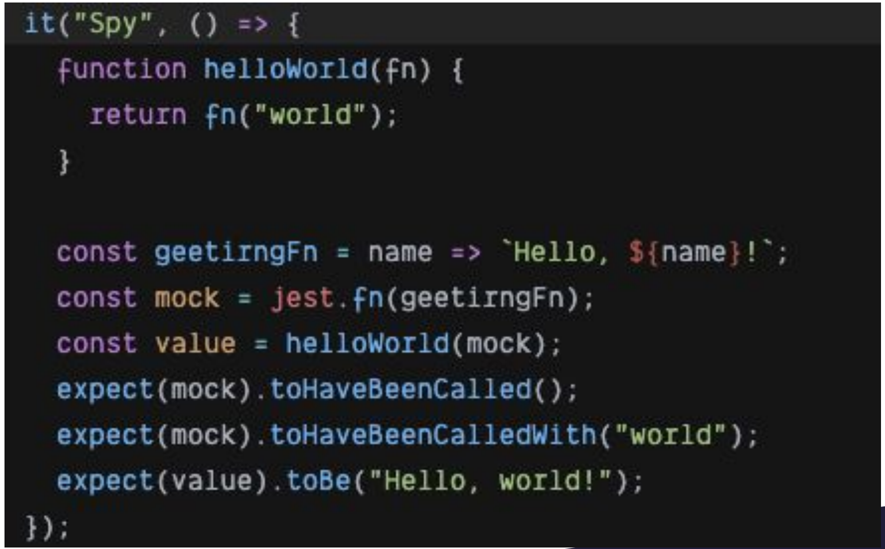
```
jest --updateSnapshot
```

Assembler
School of Software Engineering

< CODE YOUR FUTURE />

Mock Functions

We can use mock functions when we want to replace a specific return value. Or when we want to check if our test is executing a function in a certain way. We can mock a standalone function or an external module method, and we can provide a specific implementation.



```
it("Spy", () => {
  function helloWorld(fn) {
    return fn("world");
  }

  const geetirngFn = name => `Hello, ${name}!`;
  const mock = jest.fn(geetirngFn);
  const value = helloWorld(mock);
  expect(mock).toHaveBeenCalled();
  expect(mock).toHaveBeenCalledWith("world");
  expect(value).toBe("Hello, world!");
});
```

Assembler
School of Software Engineering

< CODE YOUR FUTURE />

Mock Functions

```
// users.test.js
import axios from 'axios';
import Users from './users';

jest.mock('axios');

test('should fetch users', () => {
  const users = [{name: 'Bob'}];
  const resp = {data: users};
  axios.get.mockResolvedValue(resp);

  // or you could use the following depending on your use
case:
  // axios.get.mockImplementation(() =>
Promise.resolve(resp))

  return Users.all().then(data =>
expect(data).toEqual(users));
});
```

```
// foo.js
module.exports = function () {
  // some implementation;
};

// test.js
jest.mock('../foo'); // this happens
automatically with automocking
const foo = require('../foo');

// foo is a mock function
foo.mockImplementation(() => 42);
foo();
// > 42
```

Assembler
School of Software Engineering

< CODE YOUR
FUTURE />

Let's practice a bit

https://codesandbox.io/s/friendly-cerf-ou0vk