Testing Guidelines

Version: 0
Updated: 04/08/2020
Author: Josué Silva

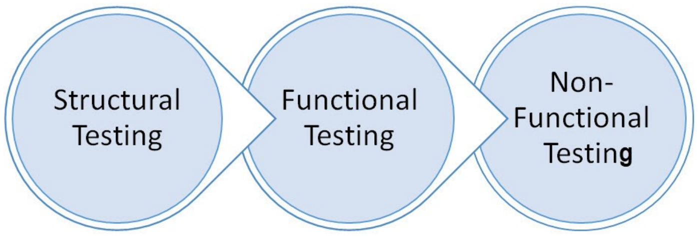
1. Test Flow

The tests have a natural flow that is divided into three parts: structural, functional and non-functional.

Structural: Focus on the code (coverage, style, quality).
Functional: Focus on the result (if the tests passed, then it works).
Nonfunctional: Focus on the unknown, looking for failures and improvement.

The scope of the Dev team remains in structural and functional tests, with non-functional tests being in charge of the quality sector.

We must focus and spend more energy on structural tests and guarantee delivery requirements with functional tests.



2. Unit tests (Structural)

Unit testing is the practice of testing individual units or components of an application in order to validate that each of these units is functioning correctly. Generally, a unit should be a small part of the application.

To achieve better results, they must:
- be reliable;
- be fast;
- be simple;
- have good maintainability and legibility;
- must verify a single use case;
- must be isolated (mocks and / or stubs must be used);
- must be automated.

3. Functional tests (Functional/API/Acceptance)

Functional testing is a type of software testing that validates the software system against functional requirements / specifications. The purpose of functional testing is to test each function of the software application, providing the appropriate input, checking the output against the functional requirements.

For best results:

- Identify and clarify the functions you expect the software to perform;
- Create input data based on these functional specifications;
- Determine the output based on these functional specifications;
- Write and execute test cases;
- Compare the results of actual and expected production.

When using an API, we need to test each feature and validate its functionality.

4. Code Coverage

Code coverage represents the percentage of code covered by testing. We must have at least 95% coverage of the code, a value below this prevents the delivery of the product.

5. C.R.A.P.

```
CRAP(m) = comp(m)^2 * (1 – cov(m)/100)^3 + comp(m)
```

"The C.R.A.P. (Change Risk Analysis and Predictions) index is designed to analyze and predict the amount of effort, pain, and time required to maintain an existing body of code. If CRAP1(m) > 30, we consider the method to be CRAPpy." - Alberto Saviola

The CRAP index should always be analyzed as it is an indicator of low code maintainability. Although its creator argues that below the number 30 is acceptable, we must always keep in mind that this value should be as low as possible.

This index is found in the code coverage report.

6. Test naming conventions

Naming standards are important because they explicitly express the intention of the test. To facilitate the reading and understanding of the purpose of the tests, we adopted the technique:

Should_ExpectedBehavior_When_StateUnderTest.

Your test name should consist of four parts:

1. Should (fixed);

2. Expected behavior when the scenario is invoked;
3. When (fixed);
4. The state under test.

Examples:
- Should_ReturnLastInsertedId_When_POSTNewContent ();
- Should_ReturnAListOfContents_When_GETAllContents ();
- Should_ReturnASpecificContents_When_GETAnIdOfContents ();