Software testing

Introduction to software testing

As we saw in earlier sections of this course, one of the stages in the software development process is testing. An it's a really important stage. In this video you can see how important is to set up an appropriate and efficient test stage.

Software tests consist in the dynamic verification of the behavior of a program, with a properly selected set of test cases. Tests are performed in order to find possible bugs in the implementation, quality or usability of a given software.

1. Targets and principles of software testing

The main targets of software testing are:

- Detect software failures or bugs, and make sure that every previously detected bug has been fixed.
- Verify the appropriate integration of the components.
- Verify that every requirement has been implemented.

In order to reach these targets, software testing must follow a set of principles:

- Test can help us find bugs, but not their absence.
- The most difficult part of the testing process is to decide when to stop.
- Try to avoid test cases that are not previously planned, not reusable and/or trivial, unless the program is really easy to test.
- Test cases must be written for both valid and not valid or unexpected input.
- The number of bugs to be found is directly proportional to the number of bugs already found. In other words, the more bugs we have detected with our test cases, the more bugs are waiting for us.

2. Stages of software testing

The stages of every testing process are:

1. Select what the test must detect, its main target.
2. Decide which kind of test is going to be performed and what kind of elements do we need to do it.
3. Implement the test cases. A *test case* is a set of data or input conditions that will be used in order to reveal something about the program, or the attribute(s) that we are checking.
4. Determine the expected results of the test cases and create a document with all of them.
5. Run the test cases

2.1. Evaluating the results

Evaluating the results consists in comparing the actual test results with the expected results. Every difference between them means that there is a bug, and this bug is usually due to the unit or attribute that we are testing, although sometimes it can be due to the test process itself, if it hasn't been properly run.

3. Test types

There are different types of tests. Let's see them from lower to higher level.

3.1. Unit tests

Unit tests check the appropriate behavior of one code unit. A code unit is typically a class in object oriented languages (Java, C#...), or a set of functions closely related in non object-oriented languages (Python, JavaScript...). They are usually run by the development team, and they must be:

- Automatable
- Complete
- Repeatable
- Independent (a unit test must not affect the result of another one)

3.2. Integration tests

They try to find bugs in the interface connections and/or in the interaction between different components or units of an application. In other words, once the unit tests are successful, we try to join all (or some of) these units and see how the program works. They are performed by the development team by applying some of these techniques:

- *Big bang*: it consists in integrating and testing everything at once (not recommended, unless the project is too simple)
- *Top down*: components are tested according to their hierarchy, from top to down. This way, bottom components that are not implemented or tested yet are replaced by auxiliary components that simulate their behavior. So, interfaces between components are checked in early stages of the project.
- *Bottom up*: bottom componentes are firstly implemented and tested, so we don't need any auxiliary component to replace them. As they are tested, then upper components can be integrated and tested as well.
- *Combined*: some parts are tested using a top down technique, and some others use a bottom up.

In this category we can also talk about **regression tests**. They consist in testing a given component whenever it has been modified, in order to find out any fault that was not previously checked. These faults can be found in either the modified code or in any other component integrated or related with the one that has been modified.

CI/CD

CI/CD stands for *Continuous Integration / Continuous Deployment*. It goes one step further after continuous integration and, if all the tests have been passed, then the application is automatically deployed on its target environment. Actually, we can distinguish between:

- *Continuous Integration / Continuous Delivery* if the deployment process after the CI is manual
- *Continuous Integration / Continuous Deployment* if this deployment process is automatic after the CI stage

We are not going to delve into these aspects here, since they depend on the project architecture itself. CI/CD can be completely different depending on the operating system, IDE and language that we are using, so there are no commong guidelines that we can learn in this stage.

3.3. Upper level tests

After the unit and integration stages, we can still differentiate two additional types of tests:

- **Acceptance or validation tests**: these tests are performed by customers and project managers in order to check that every requirement registered in the analysis stage is being satisfied.
- **System tests**: they must prove that the deployment of the application in its real environment is successful, and its behavior is as expected. In this test, the customer is also involved, along with the project manager or the development team.

4. Test cases

When we want to do any test over an application, we need to design the *test cases*. As we have said before, they are a set of conditions that can determine if software runs properly or not. The concrete definition according to the ISTQB (International Software Testing Qualifications Board) is: "*a set of input values, execution preconditions, expected results and execution postconditions, developed for a particular objective or test condition, such as to exercise a particular program path or to verify compliance with a specific requirement*".

There are several formats for these test cases, but we must include the following data anyway:

- **Identifier**: it can be numeric or alphanumeric, and it help us identify each test case
- **Name**: a descriptive (meaningful) name.
- **Preconditions**: what needs to be ready before starting the test, such as a given input file, the results of other test cases previously run...
- **Steps**: it defines the interaction with the user, such as entering a name, or pressing a button.
- **Test data**: data to be used in the test case, such as a concrete user name, password...
- **Expected result**: what the test must produce or output
- **Actual result**: result that we actually get when we run the test. This last field is filled when we run the test, whereas the rest of fields must be specified before running the test.

Example:

We want to check the behavior of a given form to log in an application. These are some of the test cases that we could specify:

ID	Name	Preconditions	Steps	Data	Expected result	Actual result
U1	ValidInput1	User **pepe** exists with password 1234	Type user and password	pepe 1234	OK	
U2	UserNotValid	User **pepito** does not exist	Type user and password	pepito 1234	error	
U3	PasswordNotValid	User **pepe** exists with password different than 4567	Type user and password	pepe 4567	error	
...						

Exercise 1:

> Design test cases to check the behavior of a function that returns a boolean indicating if the number specified as a parameter is even or not.