Level 1

Introduction

1.1 - What is automated testing?

To understand what automated testing is, we must understand first what testing is. Testing is defined as the process of evaluating a system or its components to validate whether it satisfies the specified requirements or not. It is about executing a system to find differences, errors or requirements that are not met.

Automation allows the simulation of users using the application in order to find mistakes while making validations.

It should be mentioned that automation does not replace manual testing, but rather complements it, helping to improve coverage and executing repetitive test cases, which allows manual testers to have more time to try breaking the application.

An example of how useful automated testing is can be found when we have applications that over time have grown enough, or that will grow enough, which causes manual testing to require some time and resources that make it unfeasible. Let's see the case of an application that we created from scratch; we can say that it can be tested by a single person, given that the functionalities are going to be relatively few.

As time passes, regressions (which are the tests we perform to validate that everything continues to work perfectly) tend to grow exponentially, especially if we consider that each regression should be run in a set of different systems, such as the main browsers.

Another use case is when we need a quick feedback of the application under testing, being able to integrate our automated testings with a continuous-integration system, which automatically runs the tests in several browsers while returning the errors found, if any.

1.2 - When automating is a good idea?

To understand when it is a good idea to automate, we must understand when it is not:

It is not a good idea to automate when we have a system that is under constant change, for example, when we have applications that we think are going to be redesigned, or when we simply have applications that are a conceptual draft and not a final version. In these cases, maintenance of the automated testing suites becomes very expensive and we will not obtain reliable results.

It is not a good idea either to automate when we need to perform non-functional validations or validations that need a human, such as colors or usability.

To automate an application we need a system as stable as possible, this is also why a sprint is usually automated behind the current one, to avoid a system that can change due to an error, a requiremet, or a change.

So, in addition, it's a good idea to automate when we have:

- An application that guarantees some stability
- Tests that must be run every time the application under testing is modified.
- Tests that use multiple data values for the same cases
- Tests that require detailed information of the internal states of the system (SQL, GUI, etc.)
- Stress or load tests (Performance)
- Test that must be run in several environments

We can say that it is not a good idea to automate when we have:

- Usability test
- Tests that will be run only once
- Tests without predictable results
- Unstable applications

Manual Testing vs. Automated Testing

Manual	Automated
Success depends on the expertise of the performer	The results do not depend on the performer.
Difficult to document	Self-documented
Difficult to keep updated	Authomatically updated
Slower and tedious	Faster
Non-reusable	Reusable

It should be noted that the cost/benefit of automation takes a little longer to be seen, because of the time involved in creating the automation framework plus the time it takes to create each test. In spite of this, as time goes by and the test case suites grow, we can run many cases in considerably less time than with manual testing, being able to show a positive and maintainable return on investment at low cost.

1.3 - Automated testing strategies

There are different automated testing strategies, each of them with its advantages and disadvantages. Let's see some of them:

Record and Playback: QTP, Selenium IDE, SIKULI, Eggplant:

This strategy is based simply on recording the actions performed by the user, either by images, elements or coordinates, validating a result, and then reproducing them. This is suitable when those who automate are non-technical people or when we need to create a quick test; however, this approach is very unwise for its high level of maintenance. As an example, if we have N test cases recorded, and the login of the page changes, we have to re-record the N cases.

Data-driven testing

Data-Driven Test is a strategy implemented to consume the test data used by scripts, from an external data source (XLS file, XML, database, etc.). These data specify the input and output values (verifiable, known in advance). They can also contain configuration values for the system being tested. These data are usually arranged in table format. Where different columns suggest different input/output data and where each existing row describes a new execution of the test to be performed.

Keyword-driven: Robot, Fitness

- It is an automatic testing strategy that separates the creation process in two stages:
  - Planning
  - Implementation
- This Framework requires developing a data and keywords table, independent from the automation tool used.
- Keyword-driven tests are similar to manual testing cases.
- In a keyword-driven test, the functionality of the application to be tested is documented in the table as the steps of each test.

Summary of the level

At this introductory level we saw the basics of automated testing, we marked the differences of manual testing, we reviewed the cost/benefit of automating our tests and we presented different automated testing strategies.