USING THE AUTOMATION FRAMEWORK IN TESTING SME's SOFTWARE APPLICATIONS

Lecturer Stancu Ana-Maria Ramona , PhD
"Spiru Haret" University of Bucharest, Romania

Assoc. prof. Cristescu Marian Pompiliu, PhD
"Lucian Blaga" University of Sibiu, Romania

Mara Dumitru-Alexandru
"Lucian Blaga" University of Sibiu, Romania

***Abstract**: This paper presents a way to test applications using the Python language. The first part describes the testing activity, in general terms, focusing on manual and automated testing. The second section offers information on the manual and automated testing methods that are conducted on the basis of the system. The following is an overview of the architecture and technologies used in the construction of the automation framework, underlining: Python, PyCharm, Selenium and also presenting the logic that underlies its design. In addition, it explains how to implement and test the application, describes the steps taken and the outcomes achieved.*

***Keywords:** architecture, automation, model, design, software, technology*

***JEL classification:** O35, O30, O47*

1. Introduction

In terms of applications in recent years and the development of technologies, they are increasingly accessible to users. As the number of online users has increased from year to year, software manufacturers are striving to satisfy customers and avoid their loss as well as the massive loss of money. Software applications are manually tested and validated before being put into production.

Regarding automated testing, it also has several advantages, namely: cost efficiency, reduced execution time, extension and improvement of workflow, accuracy in operations, continuous testing, etc.

In addition to advantages, we have several disadvantages, namely: it requires advanced technical knowledge to write automated test scripts, code repair can be complicated, maintenance of test data can become complicated, etc.

Software testing is an investigation carried out to provide interested parties with information on the quality of the software product or service tested. Software testing can also provide an objective and independent view of the software to enable the company to understand and understand the risks of software implementation. Testing techniques include running a program or application with a view to finding software bugs (errors or other defects) and checking if the software is suitable for use (Wikipedia).

Regarding testing, there are several approaches, namely: White-box testing, Black-box testing, Gray-box testing and four levels of testing, ie: Unit test, Integration Testing, System Testing, Acceptance Testing.

2. Literature review

In recent years, the development of new technologies and applications has become increasingly accessible to users. As the number of online users increases significantly from year to year, software manufacturers are making considerable efforts to bring software products to the highest level of quality, but also to satisfy their customers in their portfolio and avoid the risk of massive losses of money or customers.

Software applications are manually tested and validated before being put into production, as the technology used to develop them advances, they will become increasingly complex to test and validate.

The advantages of automated testing:

- *Cost efficiency*. Many manufacturers consider automation to be a strategic investment rather than a cost.
- *Reduces execution time* by simplifying daily activities and letting machines and software do the work for the user.
- *Expands and improves workflow*. Reducing business costs and the time involved in carrying out operational activities leads to improved workflow efficiency.
- *Accuracy and consistency in operations*. The potential for error is a major disadvantage associated with manual processing.

The disadvantages of automated testing:

- Advanced technical knowledge is required to write automated test scripts.
- Repairing the code can be complicated, as a minor error can lead to major malfunctions of the self-test script.
- Script maintenance can be costly if the automation framework is not made scalable for large systems and does not follow certain "design patterns" recommended by experts in the field.
- Maintenance of test data can become complicated.
- Software testing is an investigation performed to provide interested parties with information about the quality of the tested software product or service. Software testing can also provide an objective and independent view of the software to enable the company to appreciate and understand the risks of software implementation.
- It is used to evaluate software quality and should be managed throughout product development; is a process of verification and validation, ensuring that the product works in accordance with customer specifications and requirements. (Zafar 2012)
- Manual testing of the software is performed by a man sitting in front of a computer, browsing the application, trying different uses and input combinations, and comparing the results with the expected results.
- The testers follow a test plan, and perform the tests according to the detailed test cases assigned to them. The results are then collected from all testers, and a report is generated with the tests performed by them.
- Regarding the type of testing, we can talk about three approaches, namely:
- White-box testing. There are three approaches to software testing: white-box, black-box, and gray-box testing. White-box testing, also known as glass boxing or structural testing, means that the tester has a complete understanding of what is happening in the program, and the test cases are designed using programming

knowledge and skills. The tester chooses his inputs in a way that he can check all the paths through the code, and determine if the results are the right ones. (Zafar 2012) (Wikipedia Software Testing)

- Black-box testing. Black box testing, also known as functional testing, treats the program like a black box. The tester has no knowledge of how the code is executed inside the program or the logical procedures, the tester only knows what the program requires to do, and not how it does it. The tester takes over the role of end user, where it tries different inputs and checks if the program returns properly from the outputs. (Zafar 2012) (Wikipedia Software Testing)
- Gray-box testing. Gray-box testing is a combination of black-box and white-box testing (Guru99, 2009). In this case, the tester can have access to the internal code of the program and has some knowledge about the functions of the program. Testers can manipulate, for example, data inside a database, which would not be possible at the black-box testing level. Test cases are designed using this knowledge, however, all tests are still done at the black-box level.

3. Description of the framework

Automated testing can be implemented if the same test is run frequently in the same application or in certain sections of the application. It is almost impossible to test all types of tests. For example, testing the usability of the product, and yet if the workload and costs are too high for manual testing, automated testing may be considered.

Testing can be classified on several levels depending on the size, depth and object of the tests. There are four main levels of testing:

- *Unit test* - unit testing is, according to (Etheredge, 2009), the first level of testing and means testing an individual unit of code or a group of units, where a unit can be a simple method, a module or a class;
- *Integration Testing* - integration testing focuses on testing the interaction between a group of system units. While unit testing ensures that one or more methods, classes, or modules work properly individually, integration testing ensures that all components work and interact properly together;
- *System Testing* - the purpose of system testing is to verify that the software works the same when it is a fully integrated system. The software should be completely installed on different platforms with different configurations, to ensure that all scenarios are covered; (*Wikipedia Software Testing*)
- Acceptance Testing – acceptance testing is the last level of testing where, in general, the functionality of the software is tested if it meets the customer's requirements. Generally, acceptance testing is done by the customer, usually on their system or systems, also known as "user acceptance testing" to verify that the product meets the requirements before being released into production.

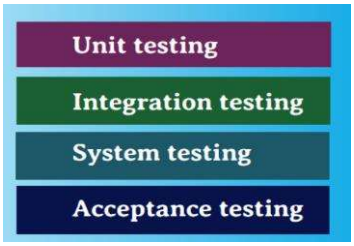

Figure 1 – Levels of software testing
Source: (adapted after (3. Educba))

4. Architecture and technologies used to build the automation framework

The first part describes the technologies used to build the automation framework, and these are:

a) **Python** is an interpreted, interactive, object-oriented programming language. Incorporates modules, exceptions, dynamic typing, very high level dynamic data types and classes. Python combines remarkable power with very clear syntax. It has interfaces to numerous systems calls and libraries, as well as to various window systems, and is extensible in C or C ++. It is also usable as an extension language for applications that need a programmable interface. Finally, Python is portable: it runs on several variants: Unix, Mac and Windows 2000, and later versions.

b) ***PyCharm*** is an integrated development environment (IDE) used in programming, especially for the Python language (Mindfire Solutions, 2018). It is developed by the Czech company JetBrains. It offers code analysis, a graphical debugger, an integrated unit tester, integration with version control systems (VCSes) and supports web development with Django, as well as Data Science with Anaconda. PyCharm is compatible with multiple platforms, such as Windows, macOS and Linux.

c) ***Selenium*** is a suite of tools for automating web browsers (Guru99, 2009). It is used, especially with different testing frameworks for automated testing of web applications, but can be used for example for automated web administration tasks. It also has support for multiple platforms, such as Windows, OS X, Linux, and Solaris, and can be controlled by many programming languages (C #, Java, Perl, Python, etc.) and automation frameworks (Bromine, Junit, Nunit, Rspec, Robot Framework).

The second part describes the architecture of the automation framework. For this, I aim to create a Hybrid Automation Framework based on the design pattern "Page Object Model" and the development principles "K.I.S.S", "YAGNI" (Nguyen Tuyen, 2019).

With the help of the automation framework I will be able to perform the following activities:

- Generating HTML reports with relevant data following the tests performed;
- Reuse of test functions and methods;
- Quality assurance on several web platforms (Chrome, Firefox);
- Providing functional tests in various sections of the application, etc.

a) Logic flowchart of the automation framework

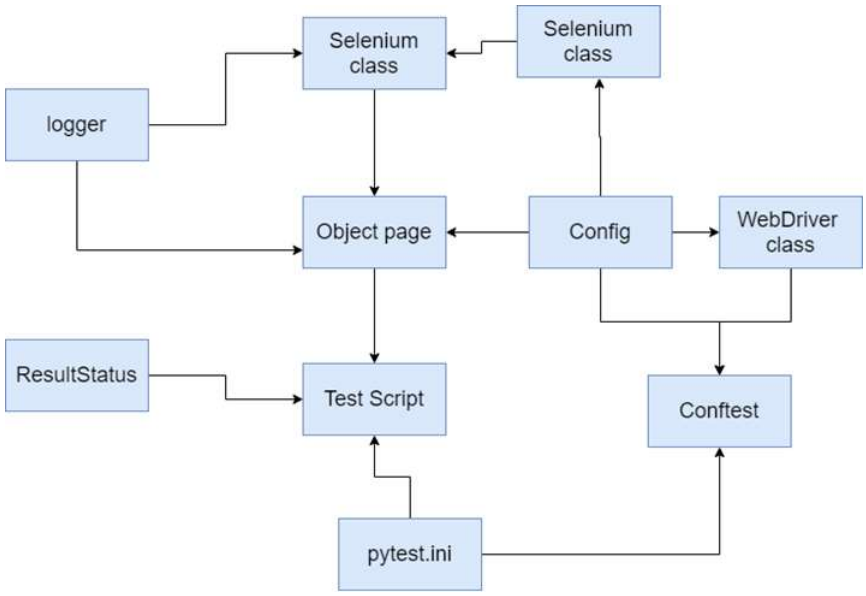

Figure 2 - Logic flowchart of the automation framework

As you can see in the figure above, with the help of the Config class I will be able to make most of the framework configurations from a single location (the location where we generate the html report, pictures, driverlorchrome or firefox location, etc.), and to modify what identification pattern we want for pytest to identify the tests, i will do it from the pytest.ini file.

b) In addition to the logic diagram, we must make two folders, one "base" in which will be implemented the classes that contain the methods that will be used and reused in the application and the other folder entitled "utilities in base", folder in which they will be implemented two classes, namely the ResultStatus class and the customLogger class, classes that will take care of recording actions and evaluating my tests if they meet the requirements or not.

c) Custom logger. In this part I will implement a function that will help me with recording all the steps taken by the test my scripts, in a file called "automation.log".

d) Creating a class that validates the tests performed. There are cases where we would like to check several steps in an automated test. Since an "assert" would stop the script at the first step which will be evaluated as "failed", we will have to develop a separate functionality, or import an existing plugin.

5. Application implementation and testing

In order to implement and test the application, you need to perform several steps, namely: to install and configure a virtual environment using PyCharm IDE, to add Firefox drivers, to create a file to install packages, to set patterns for to be able to identify the scripts using Pytest.ini, to perform the configuration and implicitly the configuration class of the Firefox drivers, to create a function that can transmit the tests to the external data and to create two classes, one Object from the web page and another one that contains the automated tests.

- Installing and configuring a virtual environment using PyCharm IDE (Jeffries, 2001). I will first create a virtual environment in PyCharm IDE that will help

me keep packages and libraries in my framework separate from other projects to prevent possible incompatibilities;

- Adding Firefox / Chrome drivers, I will create a folder called "drivers" where I will add the two drivers (chromedriver.exe and geckodriver.exe). By creating this folder I will no longer have to add drivers to windowspath.
- Create a package installation file. In this part I will create a "requirements.txt" file, where I will pass all the dependencies and packages that the framework needs to work. With a simple command (pipinstall -r requirements.txt) all the necessary packages for the framework will be installed automatically.
- Establish the pattern for identifying scripts using Pytest.ini. Pytest can identify the scripts that it wants to run according to the following pattern in the configuration file of the pytest.ini framework. This model is useful because with this model, pytest can identify the test files I want to run even if I have test files organized in multiple subcategories of folders.
- Realization of Config and implicitly the configuration class of Firefox drivers. In the first part I will implement a configuration class of my automation framework, which will allow me to do most configurations in one place, and in the second part I configure the drivers for Firefox, because it helps me not to change every time test class.
- Creating a function that can transmit data to tests used from external data. In this part I will implement a function that helps me to read external data sources that I will be able to transmit to the self-test scripts.
- Creating two classes, one Object from the web page and another that contains automated tests. In this part I will implement the first object class in the framework. This class will contain all the actions needed to search the google search engine, and in the second part after I finish implementing the object page I will create the test page that will run various tests and scenarios on that page, and in the second part.

6. Obtained results

In this section I will choose a web page and start creating automated tests using the built framework. I will use the "core" methods in the framework to apply the POM "design pattern" and I will also show how the tests that are specifically marked are executed and the tests are executed in a desired order.

Regarding the execution of tests that are specifically marked, from a certain version of the pytest, the names of all markers used must be declared in that file. It is a way to add security and reduce the risk of an overly complex automation framework. For example, if I had a few hundred tests and no specific file in which to know how those tests are marked, it will most likely increase the maintenance time of the tests (the one who performs the tests, must identify each test class and evade markers of that class, to avoid the risk of a test not being run or other tests being performed by mistake).

In other words, we can use these markers, both at the class level and at the method level, which gives me the ability to run an entire class of tests, specifying only the class marker, or to run a specific test in that class.

At the same time, pytest will give me the opportunity to mark the order of the tests and I will also be able to use other types of "marking" of useful tests such as skip or xfail.

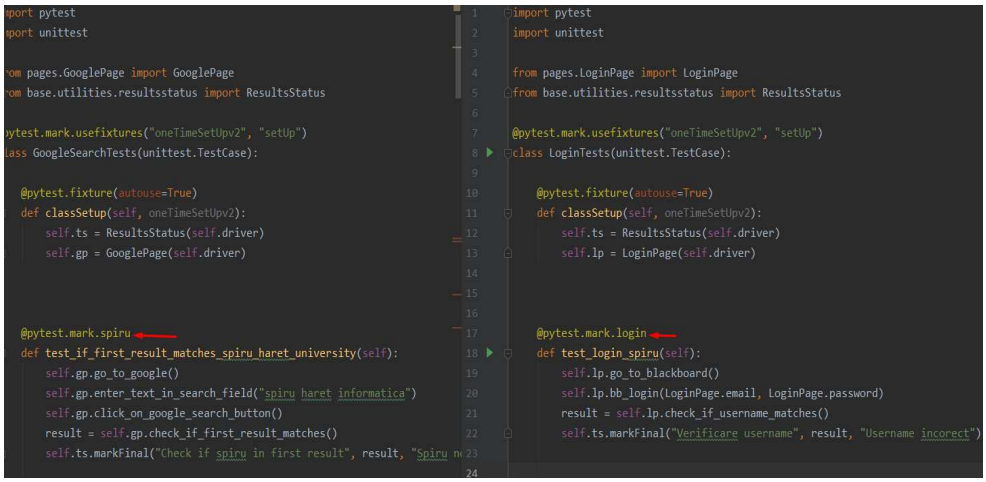

Figure 3 - Implementation of two separate tests with 2 different markers specific for the type of test

As can be seen in Figure 3, we have two separate tests belonging to separate object classes. The first belongs to the GoogleSearch class, and the second to the LoginPage class.

We want to have this format to keep the tree pattern design. In the first class of tests we will test the scenarios related to the google search engine, and in the second, we will test the scenarios related to the authentication on the blackboard of the faculty. As we mentioned before, we can mark the tests with specific markers to give me the flexibility to run them independently or together.

Conclusions

The main purpose of this work was to build an automation framework that could meet the following requirements:

a) With this framework, the test time of the application can be reduced by running the tests individually or in parallel. It can be seen that all tests can be carried out in a time interval of about 72 seconds. Manually testing these tests would take at least a couple of minutes;

b) The total execution time can be reduced to 38 seconds, if I use 4 browsers to execute them;

c) After the tests have been carried out, the framework will generate an html report with the results obtained;

d) In the event of an exception during execution, I will have a log generated automatically by the framework to make debugging easier;

e) The functions and methods that have been created can be reused in the test script, but also in other object classes;

f) The reduction of hard-coded data within the test scripts has been reduced because they can use external data sources, the functions and methods are reused, and any basic configurations can be made from the config class;

g) They may extend the testing of possible new functionalities by reusing the methods already established;

h) Tests can also be run on multiple platforms (Chrome, Firefox).

The main purpose of this work was to create a framework that could help me create automated tests for various web projects, but also to be able to expand it in the future with a variety of functionalities, to run tests on multiple OS platforms (Windows, MacOS, Mobile, Linux) or browsers (Edge, Firefox, Chrome, Opera , Safari, etc.).

With the help of this framework, we can create automated tests quickly and easily, but I can also configure them very easily with the help of commands that can take data from the terminal user input, but also with their specific configuration files. You can also create multiple order sets for future projects and needs.

REFERENCES

1. EDUCBA, (2020), *Levels of Software Testing,* [online] [2020-07-14] accessible: https://www.educba.com/levels-of-software-testing/.
2. ETHEREDGE JUSTIN, (2009),*What is Unit Testing?*, Simplethread, [online] [2020-06-29] , accessible: https://www.simplethread.com/what-is-unit-testing/.
3. GURU99, (2009), *What is Smoke Testing? How to do with EXAMPLES*, [online] [2020-05-25], accessible: https://www.guru99.com/smoke-testing.html.
4. GURU99, (2009), *What is Selenium WebDriver? Difference with RC*, [online] [2020-04-26], accessible: https://www.guru99.com/introduction-webdriver-comparison-selenium-rc.html.
5. MINDFIRESOLUTIONs, (2018), *What is PyCharm IDE?,* [online] [2020-08-27], accessible: https://medium.com/@mindfiresolutions.usa/what-is-pycharm-ide-cc0735784f64.
6. NGUYEN TUYEN, *Selenium Testing Framework*, (2019), [online] [2020-08-28], accessible:https://www.automatedtestingwithtuyen.com/post/page-object-model-pattern.
7. R.E. JEFFRIES, A. ANDERSON, C. HENDRICKSON, (2001), „Extreme Programming Installed", Addison-Wesley Professional.
8. WIKIPEDIA *Software Testing*, [online] [2020-04-03] accessible: https://en.wikipedia.org/wiki/Software_testing.
9. WIKIPEDIA, *Regression testing*, [online] [2020-04-03] accessible: https://en.wikipedia.org/wiki/Regression_testing.
10. R. ZAFAR, (2012), *What is software testing? What are the different types of testing?*, [online] [2020-05-25], accessible: https://www.codeproject.com/Tips/351122/What-is-software-testing-What-are-the-different-ty.