Automated Software Testing: An Overview

Rishab V Arun[1], Ganashree K. C.[2]

[1]*B. E. student, Department of Computer Science and Engineering, R.V. College of Engineering, Bengaluru, Karnataka, India*
[2]*Assistant Professor, Department of Computer Science and Engineering, R.V. College of Engineering, Bengaluru, Karnataka, India*

---***---

Abstract - *Before a software product is made available for a client to use, it must undergo thorough testing. Software testing ensures that all the features of the software product are working fine and there are no anomalies. Software testing not only checks for errors but also makes sure that the software meets the requirements that were laid out before/during its development process. Software quality is one of the most important factors that have to be met in software development. In most companies, software builds are released regularly in a short span and it is necessary that the issues in the software product are identified and fixed early in order to reduce the development cost. Software's that undergo a proper testing process ensure that the maintenance cost is reduced, time is saved and the customer is satisfied. Catching a bug and fixing it in the early phases of development ensures reduced cost as one bug can cause a chain of bugs in the software.*

The software testing process can be conducted either manually or, the process can be automated. In manual software testing, no automation tools are used, and each test case is executed manually by a person. The test cases are designed to meet the software requirements and ensure the features are working as it was mentioned in the software requirement documents. However, there is a chance of human error in manual testing. There is no guarantee that in manual testing all the features are tested. The features that are tested depends on the tester. And testing of all the features of a software is a time-consuming and human resource-intensive process. In large organizations which have regular build releases, it is necessary that a large amount of testing is automated in order to ensure faster releases and reduced costs. This paper gives an overview of the testing automation process followed in software companies to test their software product.

Key Words: **Automation testing, Device under test (DUT), Testbed, Test suite, Test script**

1. INTRODUCTION

Software testing is one of the most important parts of the software development life cycle. A proper testing process ensures reduced cost, faster releases, and customer satisfaction. Software testing in most large companies is automated. Although all features that need to be tested in a software product cannot be automated, a large set of these features need to be automated. Software testing automation ensures the faster release of new software builds and reduces the human resources spent on the testing process.

Testing automation is carried with test scripts that are run on the software product. The test script needs to be designed according to a test case specification. Scripting languages like Python or Pearl are often used to develop test scripts. Usually, organizations have an automated test framework that runs the test scripts on different environments and records the results in a log file for analysis. The test scripts are designed to test the software in different environments. Automated testing often involves the use of test automation tools to help in the testing process. Selenium is one such test automation tool that is widely used for the automation of web-based tasks.

Consider a company that develops a software product that runs on a web browser. The product needs to undergo testing in different environments. It needs to be tested on popular web browsers like Mozilla Firefox, Google Chrome, Internet Explorer and the product should also be tested under different operating systems like Microsoft Windows, mac-OS, Linux. The manual testing of the software on each of these environments after the release of every build will consume a lot of human resources and time. Manual testing does not scale well. Testers can become fatigued after testing a large number of features which diminishes their ability to accurately identify bugs. Therefore it would be better if the company chose to automate the testing process for the software product. However, there are some disadvantages associated with automation testing. Whenever there is a major change in the third-party software like the OS or the web browser on which the software is run, the test scripts need to be updated accordingly to accommodate the change. Even when there is a change in the new build of the software being tested, there might be changes that need to be done in the test script that was working well with the previous versions of the software. Sometimes the test scripts need to be changed entirely and coded from scratch when the changes are major. For these reasons, the test scripts must be modularized to reduce the amount of code that needs to be changed and also to easily identify the scripts that need to be changed. In large organizations, there will be separate testing and development teams. It is necessary that the changes in the developed software are communicated well by the development team to the testing team so that the test scripts can be changed accordingly.

This paper talks about the automated testing process that is followed by an organization. The organization has a software product that has a regular release of new builds, hence most of the software testing process is automated. The software product is used by clients in different environments, and whenever a customer experiences an issue the software is tested under the required environment to identify the issue. Due to this tests need to be conducted frequently on the product therefore automated testing is preferred. The use of automation testing has ensured the faster release of new builds and customer satisfaction due to the reduced time taken to resolve the issue.

2. Methodology

This section talks about the process of automation testing. Automated testing is carried by a testing framework which runs the test scripts. The testing framework will run the test scripts on the software product in different environments. The results of the test will be stored in a result/log file for analysis. For better analysis of the result, the testing framework will have a GUI to display the result/log file. The management of the testing resources and the conduction of the test can be done with the GUI/CLI. Fig -1 shows the testing process flow on a high level. The test script is run by the test automation framework on a test machine which contains the software product being tested. After the completion of the test the test results are available in a result/log file.

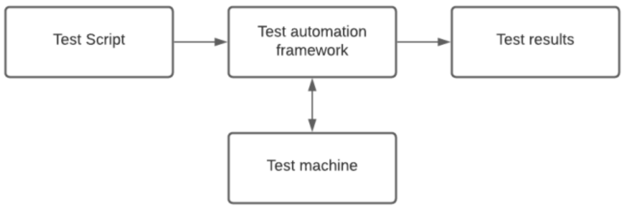


Fig -1: Test automation process

2.1 Components of an automated test system

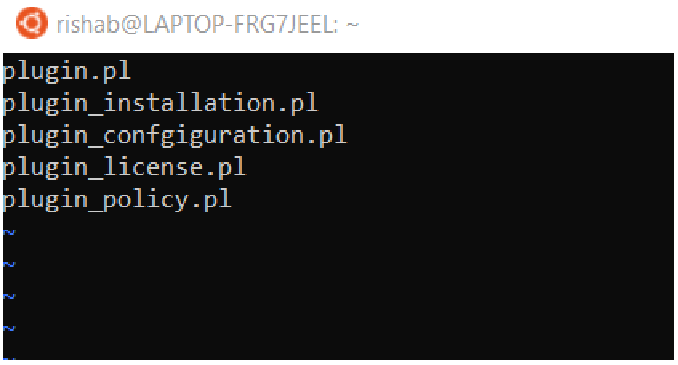
An automated test system involves many components like the test script, suite file, etc. These components are described below:

1. Test Script: A Test Script can run a single test case or set of test cases for a particular feature. Each of these test cases is usually coded as a single function. These test cases may have iterations that represent different cases of the same test case. To automate the testing of a software, a collection of scripts that test its different features are compiled into a suite. The test suite is run by the automated test system on the testbed.

A test script is usually written in Perl/Python. Every script is associated with a unique script ID which is used to identify the script. A test script may be associated with a configuration file to run the test case on devices with different configuration. The configuration file is used to configure the device under test (DUT) before the test cases are run on it. It is associated with a .cfg extension.

A testbed is a set of variables associated with test scripts that are different for each test run. For example, a testbed could include a variable set of domain names, IP addresses, usernames, passwords, etc. To write a script that is independent of a testbed, the variables used are replaced by values that are specific to a testbed such as the IP address when the script is parsed. These are generic variables. As the parser provides the abstraction layer which allows you to abstractly reference variables that are specific to a testbed, the same script can be run on different testbeds.

2. Test suite: A suite file contains a list of scripts that test various product features of a testbed. A suite file is created by selecting scripts listed under different features. For example, a suite file may contain scripts that test the server-side features of the software. It has a .stf extension and contains a list of scripts that need to run together. An example of a suite file is shown in Fig -2. It contains a set of scripts related to software plugin testing.



```
plugin.pl
plugin_installation.pl
plugin_confgiguration.pl
plugin_license.pl
plugin_policy.pl
~
~
~
~
```

Fig -2: A sample test suite file

3. Test Instance: When a Test Suite is executed, a test instance ID is created for that test instance. A test instance ID is a timestamp. Each test instance possesses a unique test instance ID which is generated when the test suite is run. This ID is necessary to identify a test job that has been executed and to obtain the result and logs of the test. For example, Test-2021-01-04-17-02-54-97 is a test instance ID that was generated by the automated test system. It includes the details of the year, month, day and the time at which the test execution started. A test job is referenced by its test instance ID. The automated test system allocates a directory to each test instance to store files like results file, log file, script files, configuration file, etc.

4. Result and log files: These files are created after a test run is initiated. They provide details of the execution of the test process. They are the final output of a test run.

A. Result file: A result file is created after the execution of a test script. A test suite consists of many scripts. Hence, the result file for each of the scripts of the test suite is stored in their respective test instance directories when a test suite is run. It provides details of the number of test cases that have passed and the number of test cases that have failed. It also includes information on the reason for failure or the outcome of a test case. They have a .res extension.

B. Log file: A log file is created during the execution of a test script. The log file contains the information on how the test script was run on the DUT. The log files for each of the test scripts in a test suite are stored in their respective test instance directories that are created when a test suite is run. They have a .log extension.

2.2 Automated Testing process

The automated test is triggered via CLI/GLI against a test suite and a testbed. The testbed needs to be free and shouldn't be allocated to some other test run. The user should also make sure the DUT is free and not running some other test. When a test run is started, the test instance and test instance directory are created. The timestamp when the test starts is used to name the test instance. The automated test system gets the suite file and the scripts within the suite file. A copy of these scripts is created in the test instance directory. The system then checks whether the DUT is reachable and the testbed is available if they aren't, then the test ends with test status as "Testbed busy" and "Testbed down", respectively. If the resources needed are available then the first script in the suite file is parsed. The resources are allocated to run that script, and the resources are mapped to the variables in the script, which may be IP addresses, domain names. While parsing the script if it is found that the resources mentioned aren't available then the test status is updated as "Parse error".

After this, if all the requirements are met then the test status is changed to running. When the test is running, it can be terminated from the CLI/GUI. A timer is started at this time to measure the time taken for the script to run. It also serves as a means to terminate the test when it takes too long to run. Before executing the script, it is first checked for syntax errors. If there are any syntax errors then that script cannot be executed. The test status is set as "Syntax error". If the syntax is correct, then the script is executed. The script may call several other scripts or libraries. When the test script starts executing, a log and a result file for this test are created in the test instance directory. The log file is updated as the test is being executed. The log file provides all the details of the test run process, which can be used to debug the test upon failure. The details include the functions called, networking packets sent, the status of resources, and all details which help the reader understand the process flow of the test case. The result file has details of the success or failure of each step of a test case. The result file is filled after the execution of the test script with the details from the log file.

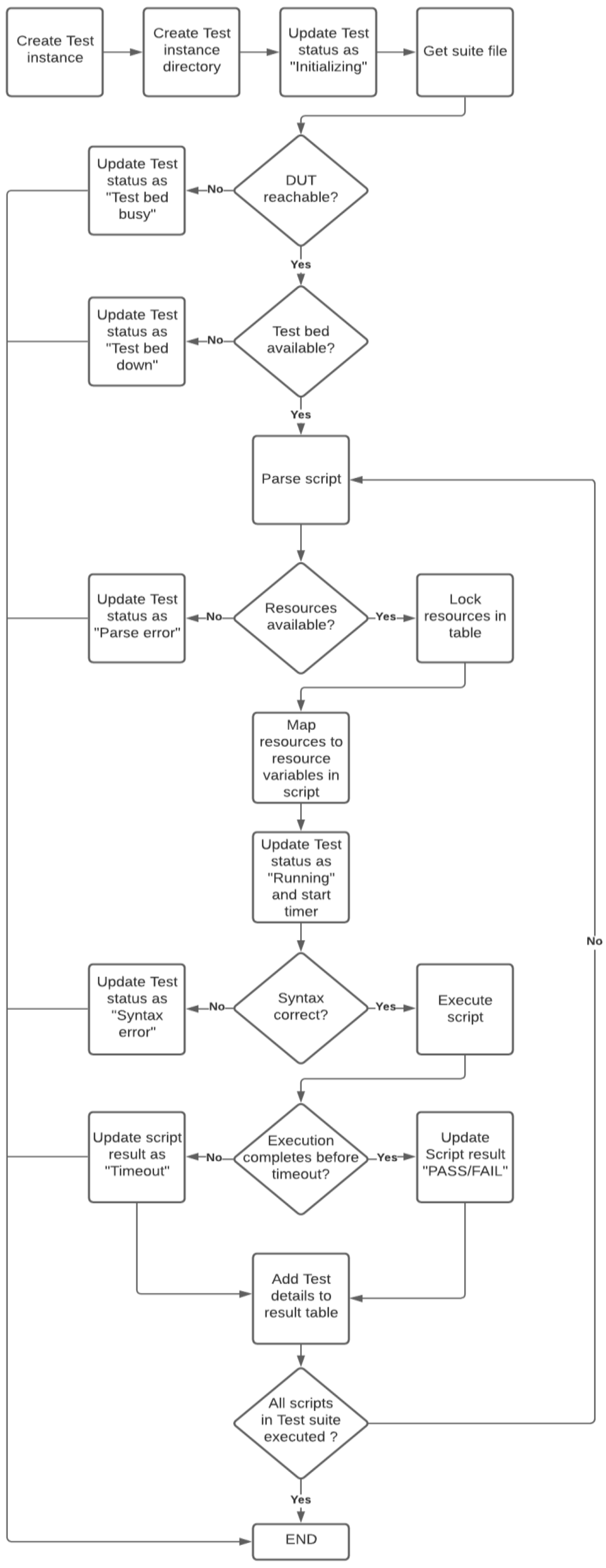

Create Test instance
Create Test instance directory
Update Test status as "Initializing"
Get suite file
DUT reachable?
No
Update Test status as "Test bed busy"
Yes
Test bed available?
No
Update Test status as "Test bed down"
Yes
Parse script
Resources available?
No
Update Test status as "Parse error"
Yes
Lock resources in table
Map resources to resource variables in script
Update Test status as "Running" and start timer
Syntax correct?
No
Update Test status as "Syntax error"
Yes
Execute script
Execution completes before timeout?
No
Update script result as "Timeout"
Yes
Update Script result "PASS/FAIL"
Add Test details to result table
All scripts in Test suite executed ?
No
Yes
END

Fig -3: Flow chart of automated testing process

If the script takes more time than expected, it needs to be terminated to avoid the system being stuck on a single script. For this reason, a maximum time for each script is assigned; this time varies for each script. If the script execution time exceeds this time then the test status is set as "Timeout".

After the successful execution of the script and the result file is updated, the final result table is updated. The final result table is displayed in the GUI and includes all the details of the test run which includes test instance ID, test execution start and end times, script files, result and log files, etc. After this, if there are more scripts to be executed in the test suite then the next script is parsed, else the test ends. Fig -3 depicts the flow chart of the automated testing process which includes all the steps taken and decisions made in the process.

ACKNOWLEDGEMENT

I would like to extend my gratitude towards my guide Prof. Ganashree K. C. for her continued support and guidance towards my work in this paper. I would also like to extend my gratitude to the Department of Computer Science and Engineering, R.V. College of Engineering for giving me the opportunity to write a paper on this topic.

REFERENCES

[1] I. Dobles, A. Martínez and C. Quesada-López, "Comparing the effort and effectiveness of automated and manual tests," 2019 14th Iberian Conference on Information Systems and Technologies (CISTI), Coimbra, Portugal, 2019, pp. 1-6, doi: 10.23919/CISTI.2019.8760848.

[2] E. H. Kim, J. C. Na and S. M. Ryoo, "Implementing an Effective Test Automation Framework," 2009 Annual IEEE International Computer Software and Applications Conference, Seattle, WA, USA, 2009, pp. 534-538, doi: 10.1109/COMPSAC.2009.188.

[3] C. Klammer and R. Ramler, "A Journey from Manual Testing to Automated Test Generation in an Industry Project," 2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C), Prague, Czech Republic, 2017, pp. 591-592, doi: 10.1109/QRS-C.2017.108.

[4] K. Sneha and G. M. Malle, "Research on software testing techniques and software automation testing tools," 2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS), Chennai, India, 2017, pp. 77-81, doi: 10.1109/ICECDS. 2017.8389562.

[5] Kassab, Mohamad and DeFranco, Joanna and Laplante, Phillip. (2016). Software Testing Practices in Industry: The State of the Practice. IEEE Software. PP. 10.1109/MS.2016.87.

[6] M. Böhme and S. Paul, "A Probabilistic Analysis of the Efficiency of Automated Software Testing," in IEEE Transactions on Software Engineering, vol. 42, no. 4, pp. 345-360, 1 April 2016, doi: 10.1109/TSE.2015.2487274.

[7] J. Gmeiner, R. Ramler and J. Haslinger, "Automated testing in the continuous delivery pipeline: A case study of an online company," 2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), Graz, Austria, 2015, pp. 1-6, doi: 10.1109/ICSTW.2015.7107423.

[8] C. Klammer, R. Ramler and H. Stummer, "Harnessing Automated Test Case Generators for GUI Testing in Industry," 2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Limassol, Cyprus, 2016, pp. 227-234, doi: 10.1109/SEAA.2016.60.

[9] S. Demeyer, A. Causevic, K. Wiklund and P. Potena, "The Next Level of Test Automation (NEXTA 2020)," 2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), Porto, Portugal, 2020, pp. xxii-xxii, doi: 10.1109/ICSTW50294.2020.00013.

[10] Umar, Mubarak Albarka and Chen, Zhanfang. (2019). ‘A Study of Automated Software Testing: Automation Tools and Frameworks’. 8. 217-225. 10.5281/zenodo.3924795.