Automated Software Testing Frameworks: A Review

Milad Hanna
Department of Computer Science, Faculty of Computers and Information, Helwan University, Egypt

Amal Elsayed Aboutabl
Associate Professor at Computer Science Department, Faculty of Computers and Information, Helwan University, Egypt

Mostafa-Sami M. Mostafa
Professor of Computer Science, HCI lab, Faculty of Computers and Information, Helwan University, Egypt

ABSTRACT
Manual software testing has been traditionally used in the software industry. It depends completely on human testers without the help of any tool to detect the unexpected behavior of an application. However, the main problem in the manual testing approach is that it is a time-consuming task in addition to the fact that tests cannot be reused. Automation software testing has been introduced to reduce testing efforts and detect as many faults as possible. Test cases are executed not only to test the functional requirements for the first time, but also to check the functions which have been already tested. This study aims to present the main features of different automation testing frameworks. In addition, an overview of different scripting techniques is presented during the study.

Keywords
Software Testing, Automated Software Testing, Test Data, Test Case, Test Script, Manual Testing, Software Under Test.

1. INTRODUCTION
There are many ways through the Software Development Life Cycle (SDLC) to control the product quality, such as careful design process management, analysis and implementation. However, software testing is the major method to control and monitor quality [1] [2] [3]. Software testing helps software programmers to fix bugs as early as possible in the SDLC to decrease the bug fixing cost [6]. This opens research on how to achieve the best possible quality in less time. The National Institute of Standards and Technology mentioned in a report that software failures cost nearly $60 billion every year [4]. Testing comes exactly before the final delivery of the final version of the software [5] [6] [7]. Software testing activities often consume from 30% to 40% of the total development costs [8] [9] [10]. Bhondokar and Ranawade [10] illustrate that during the past ten years, the software testing field has grown rapidly because applications are getting more and more complex.

Software testers face a problem in manual testing when they need to run test cases repeatedly especially if the application versions change frequently. The same problem occurs if the tester wants to run test cases over multiple browsers or multiple platforms. Therefore, manual testing is a tedious job because testers repeat testing with every change in the SUT. Moreover, manual tests cannot be reused [11] [12]. It is most suitable for non-repeatable tasks. It is usually used for revealing new and unexpected defects [13].

Automation of software testing is the process of creating a program (test script) that simulates the manual test case steps in whatever programming/scripting language [14] with the help of other external automation assisting tools [11] [15]. Since automating tests means automating the manual process which is currently in use [16], automation testing requires clear manual testing process to be able to automate it [16] [17]. Testing engineers must implement and run a program to test the SUT [18]. In other words, they implement toolkits to test the already implemented source code [11]. Automated testing is a development activity [19] which involves automating an already existing manual process.

Automation focuses on execution phase [20]. It increases the test execution speed as it can be used many times with no more effort. For sure, for the first run, it will take long time to achieve this. However, after the test scripts are ready, the human tester can execute them automatically on the SUT [21] [22]. It has a very high impact on saving the cost of the software testing phase [23].

2. AUTOMATED TESTING
A test script is a sequence of processing steps executed by the application. Each step may have parameters such as the value to be put in specific HTML control. These steps are implemented using any high level programming language [24]. Creating test scripts is a programming activity that describes test case input, output and expected behavior. Any test script is composed of three main components. The first component is responsible for starting up the SUT, the second one is responsible for exercising the main scenario steps and the last one is responsible for verification of the expected results.

Reddy [25] and Devi [26] listed detailed automation steps:

1. Automation feasibility and planning: involves discussing the scope of testing, practices to be applied and deciding whether to automate the project or not.
2. Automation design: involves selecting specific test cases to be covered in the automation since not all test cases are good candidates for automation as well as selecting an automation tool. In addition, it involves assigning automated testing tasks to the appropriate team members.
3. Test scripts development: involves implementation of test scripts that simulate test cases steps.
4. Test scripts deployment: getting the automation project ready for use.
5. Automation execution: the implemented test scripts are executed on the SUT.
6. Test verification: actual results extracted from the second step are compared against the expected results to mark every test case as either passed or failed.
7. Automation maintenance: test scripts need to be updated frequently to match any update in the source code.

3. AUTOMATION TESTING FRAMEWORKS

Historically, early automated software testing frameworks adopted the record/playback approach, then moved to the data-driven approach, and nowadays it is currently moving to be keyword-driven [27]. These approaches can be dived into two main automation testing approaches.

3.1 Record/Playback Automation

The first approach is record/playback automation testing framework is attractive particularly for non-programmers because of its ease of use. All what is needed is simply clicking the record button to record user actions, then clicking the playback button to replay the auto-generated scripts. There are many record/playback testing tools that record all user actions and data input to the different web pages of the SUT. Actions may vary such as clicking buttons, selecting values, inputting data values, etc. These auto-generated scripts are used later to run automatically without any user interaction [28] as shown in Fig. 1. Creating test cases in the record/playback approach does not require any advanced testing skills or any programming skills. Testers just need to run the web application and record their actions. However, the auto-generated test scripts are very fragile and sensitive to any simple change. Any minor change in the application GUI might break the auto-generated test script. This means that test scripts are tightly coupled to the web pages. For example, test scripts may fail to locate a hyperlink or an input field that is changed from dropdownlist to checkbox or submission button because of layout change. The solution to this problem is either to repair the test script to match the new UI change or re-record the user scenario again on the new release of the application and generate the test script from scratch.

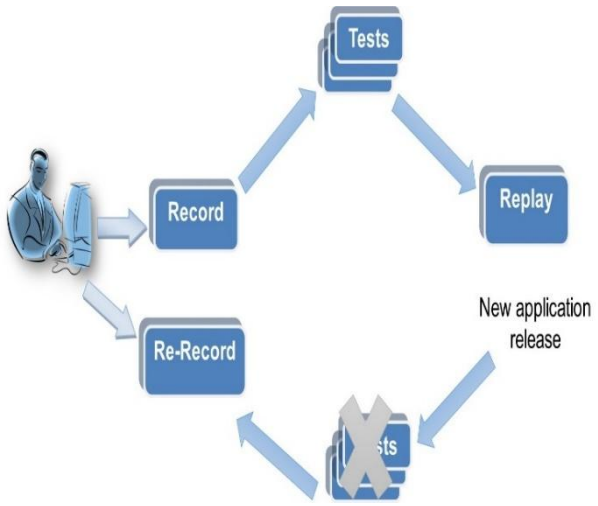

Fig. 1. Record/playback Automation Approach

3.2 Programmable Automation

The second approach is programmable automation testing framework which aims to automate applications by using all the features, guidelines and best practices of the traditional development. In this type, testing engineer can use: handling conditional execution to select one path from multi paths, loops to execute specific portion of code many times, handling exceptions and logging reuse of common methods, reference elements, parameterizing methods. It is built on the concept of encapsulation.

Nguyen and Robbins [29] named the programmable automation testing as script-based automation testing which asks testing engineers to implement test scripts to control the GUI.

Programmable approach requires elevated level programming skills because it is a normal development project, so it requires high initial effort in scripts development. The programmable test scripts that are implemented using this approach are more flexible than the scripts which are generated by record/playback tools. It is based on the manual implementation of test scripts as shown in Fig. 2.

Test scripts can be implemented using any general-purpose programming language (such as C++, Java, and Ruby). They also use specific UI libraries that can catch browser instance and provide commands that deal with HTML UI objects.

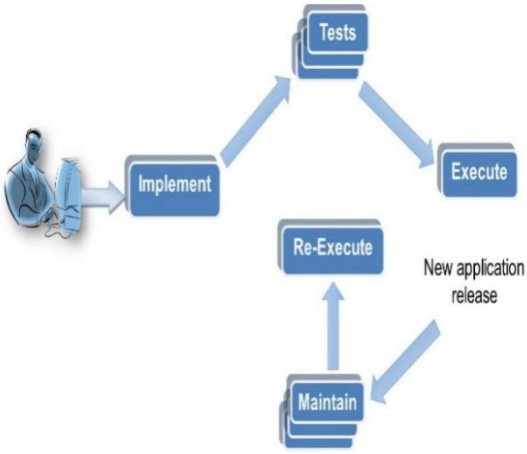

Fig. 2. Test Script Life Cycle

4. KEYWORD DRIVEN AUTOMATION TESTING FRAMEWORKS

4.1 Keyword Driven Automation Testing Framework Main Concept

Keyword-driven frameworks are based on the concept of separating not only test data but also keywords. Keywords are translated into actions using an automation driver. It is an extension of the data- driven framework where user actions are separated as keywords in addition to test data. Every keyword is related to a specific functionality, and the sequence of keywords is automatically run using a driver program. The suite of automated test cases will later run without any human intervention. It works on a higher level of abstraction [28] as it implements reusable functionalities in the form of keywords that represent test case steps [30]. The test script engine is responsible for calling the corresponding method for the appropriate keyword. Fig. 3 illustrates the high-level architecture diagram for a keyword-driven framework [31].

A keyword-driven automation framework consists of three main components [32] [33]:

- **External data files:** which consists of keywords and test data.
    - Keywords: The keywords sequence represents the test case flow. Based on these keywords, specific functions will be called.
    - Test data: includes test case inputs and outputs. Input values can either be stored with the keywords repository or separated in an external data file.

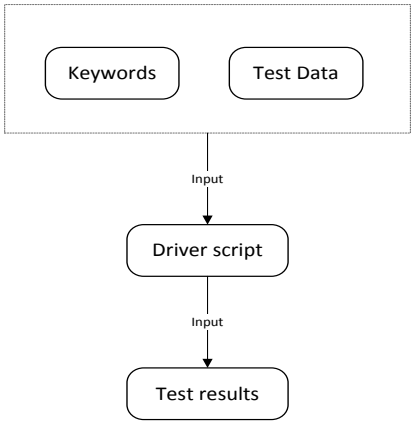

Fig. 3. General Keyword Driven Framework

- **Test function libraries**: These functions should open and read the external data source line by line and then map each keyword to its corresponding function. It is also responsible for mapping each test step to the automation source code (e.g. Selenium, Watin, QTP, etc.) that integrates with the framework.
- **Driver test scripts:** It is responsible for initiating the function library to start execution.

The main benefit of keyword-driven frameworks is that it reduces the overall cost of test scripts maintenance because of the high level of abstraction. Moreover, tests are easier to understand by inexperienced testers/users [34]. Using keyword-driven automation frameworks, the tester can create new tests without having programming knowledge.

4.2 Related Work for Testing Frameworks

Leotta et al. [35] performed an experimental analysis to calculate the cost/benefit tradeoff of the record/playback scripting approach (using Selenium IDE) and the programmable scripting approach (using Selenium Web driver). They do not only calculate the cost for creating the test scripts from scratch but also the cost for maintaining the test scripts after publishing a new release of the application. They asses the two approaches on both the short term and long term. The results of experiments on testing six different web applications indicated that [35]:

- Cost for the development of test scripts using the programming scripting technique is more expensive than using the record/playback scripting technique as it has additional time required ranging from 32% to 112% [35]. However, test scripts maintenance in the programming approach cost less than in record/playback approach as it saves from 16% to 51% [35] of the required time. They noticed that after about one to three releases of the same application, the cost of developing test scripts in the programmable approach is less than the cumulative cost of maintaining record/playback scripts. The saving cost value increases gradually after each release. This means that for any web application which is expected to have three releases over its progress, programmable test scripts will have more return on investment than record/playback test scripts.
- The more the reuse of page objects across test cases, the lower the maintenance cost needed to update test scripts because shared page objects will be maintained only once. This depends on the modularity of the web application under test.

Bhondokar et al. [27] propose usage of a hybrid testing framework which combines both data-driven and keyword-driven concepts. This type of framework can be used widely in any type of web application for automation testing as shown in Fig. 4.

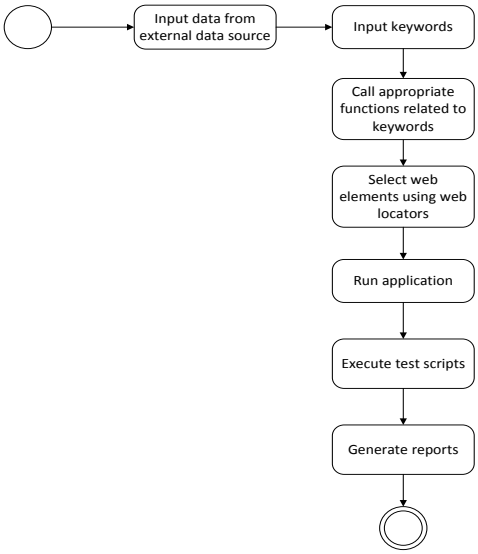

Fig. 4. Hybrid Testing Framework

Sinha [14] propose another solution to reduce the cost of automation testing by transforming the English manual written test cases to a well-organized keyword-driven form sheet. These auto-generated keywords will be used later by the automation framework to test the SUT. The main issue in this solution is that output test steps from the English written test cases are not 100% guaranteed to be correct because the user can express a test case in many different forms [36]. This requires human intervention to revise the auto-generated steps before running them. Revision of hundreds of steps is a difficult and time-consuming task. Therefore, there are recent researches that focus on detecting user intent from natural languages [37] [38] [23] [39] [40].

Lau [38] propose the Co-Tester system suggesting a new language called ClearScript. The tester should provide the co-tester system with segmented test steps so that the system can handle them. Little and Miller [40] proposed another solution to transform the tester's keywords to the user interface of a specific system.

Fei Wang [41] proposes an automated framework for testing web applications based on Selenium and JMeter. It is used for performance testing by simulating a heavy load on a server. This framework has four main components. The first component is the model which is responsible for converting each test case for object models such as elements, actions and assertions. The second component is the translator which is responsible for converting each test case into a set of actions. Then, these actions are converted into its corresponding test script. The third component is ActionWorker which is responsible for calling the testing tool to execute the actions. The last component is the comparator which is responsible for comparing the actual test results against the expected results.

Anuja [42] proposes a keyword-driven framework which is called WAT (Web-Based Automation Testing) developed in Java. It depends on generating GUIWebObjects for the web

page to be tested to perform GUI actions on these HTML web objects. Then, functional testing is performed using these GUI action-events. This framework architecture diagram is shown in Fig. 5.

WAT framework [42] consists of the following components:

- Web objects: a test case consists of test steps, each test step works on a different HTML web object such as button, dropdownlist, radio button, checkbox and tabs to perform the required test case step.

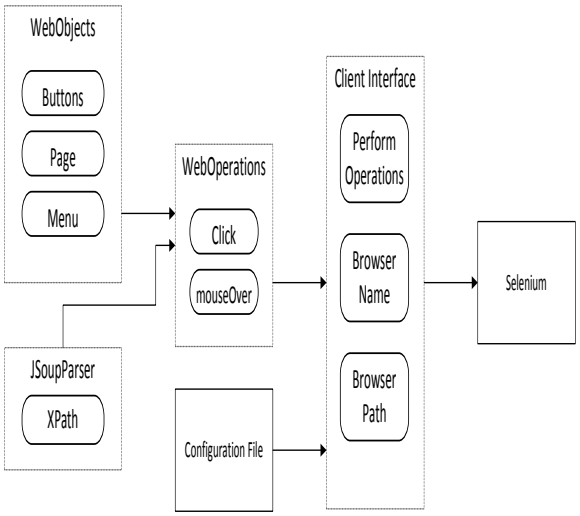

Fig. 5. Keyword Driven Framework Architecture

- JSoupParser and WebOperation: each HTML web object has a different tag such as id, name and class. Java soup parser is responsible for creating XPath for each HTML control. This XPath will be used later to locate the web element in the web page of SUT so that the automation framework can use this HTML control to achieve the business scenario such as Click, SelectValue, Type…. etc.
- Configuration File: this file is responsible for specifying the test script that will run on which web browser This file is editable so that the tester can update it to run the test script on a different browser.
- Client: It sends the commands of the test script to the selenium engine to run them on the web browser.

Verma [43] and Singla [44] propose two frameworks which are based on integrating the keyword-driven scripting technique with Selenium automation tool. Both authors propose almost the same main features and components. Keyword-driven testing simulates user actions on the SUT. It is used by testers to execute test cases and then extract the final test results. Using this framework, testers do not any programming skills. The main idea is the use of keywords which are related to functions. These functions are parameterized, so tester can update keyword parameters and create new test cases using keywords lookup. This framework integrates with Selenium. Singla [44] lists the main common components of keyword-driven automation testing framework as follows:

- Functionality class library: each functionality in the SUT has a corresponding method.
- Test data sheet: this sheet has the test data which is needed to execute the test case. The main columns for this sheet are: test case id, object type, object identifier, keyword, and data.
- Selenium web driver: To start test case execution, the code must have a web driver to initialize an instance of the web browser.
- Result reports: this report has the final testing result for the executed test cases.
- Driver script: this script is responsible for reading the keywords and mapping them to their corresponding methods to be executed on the SUT.

Ashutosh [45] propose development of a software automation testing framework for avionics system only. However, its main problem was that this model cannot be used to test different kinds of applications. Yalla and Shanbhag [46] state that the best way to reduce testing effort is mixing reusable testing framework with open source automation tool.

Stresnjak [47] demonstrates usage of Robot framework in automation testing. It is a keyword driven framework which manipulates test cases which is stored in external source. Used keywords should be implemented in test libraries to be executed on SUT. Pajunen [34] describes the Robot framework as a generic keyword-driven software testing automation framework. Test cases are composed of higher level user keywords which are composed of lower level keywords which are translated to test scripts. First, lower level keywords are packed together in framework libraries. Then, the tester can create new user keywords by using different combinations of the lower level keywords. After that, the test cases will be executed on the SUT.

Madhavan [48] propose a semi-automated keyword driven automation framework called “Autotestbot” to be used in the acceptance testing phase. However, it is tightly coupled with Selenium automation tool in Firefox web browser only. Fig. 6 shows the framework architecture [48] with its main components:

- Test cases repository: The framework needs a repository of acceptance test cases from similar application types as input. These test cases are then manipulated using the framework NLP engine. This repository will be the knowledge base for the proposed framework during execution.

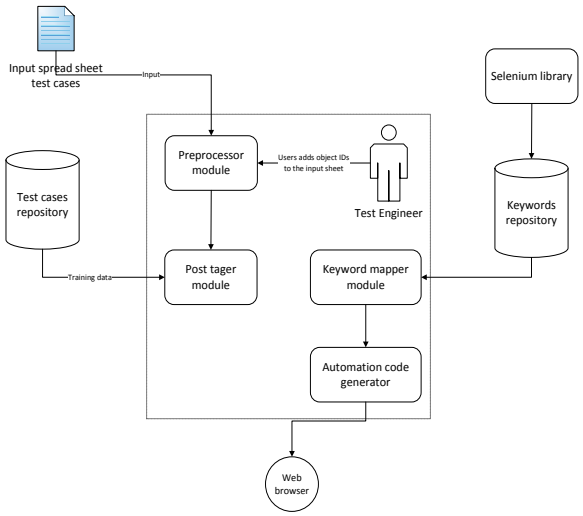

Fig. 6. Semi-automated Keyword Driven Automation Framework

- Keywords repository: Mapping is created between actions in the test cases steps and their appropriate selenium web driver keywords. Thus, each action will be mapped to selenium keyword in a dictionary.
- Preprocessing module: This module has the responsibility of reading the input test cases and uses existing tool kit for text preprocessing operations.
- POS tagger module: This module is responsible for reading the processed test cases and assigning parts of the selected tags to these tokenized test cases using the test cases repository.
- Keyword mapper module: This module is responsible for selecting the corresponding Selenium action that matches the test case step.
- Code generator module: This module is responsible for generating the test script in Python. A test case is mapped to Python test method that works upon Selenium web driver.

Castro et al. [49] propose an extension to Selenium automation tool, called "SeleniumDB". This extension adds new functionality to the assertion statements by adding the feature of connecting to database to check whether data is saved successfully or not as shown in the Fig. 7.

The results of applying this extension was saving 88% of the total time spent in executing test cases compared to the semi-automated approach and 92% compared to the manual approach [49].

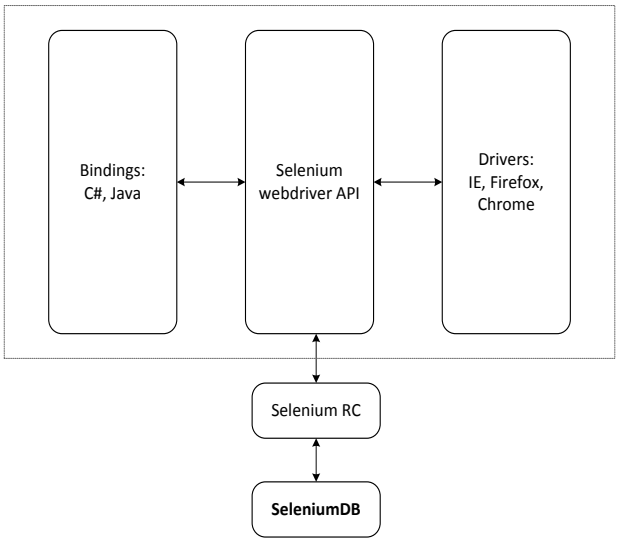

Fig. 7. SeleniumDB Extension to Selenium

5. CONCLUSION AND FUTURE WORK

Maintenance of application is always required to resolve defects, add new features, and enhance existing features. Therefore, regression testing is an important software testing phase that it executed after every application change [50] [51] especially for large scale applications that are frequently updated. However, regression testing consumes large amounts of time as well as effort because it requires re-running test cases which were already executed [52]. Chittimalli et al. [53] mentioned that regression testing consumes about 80% of the total software testing estimated budget. For these reasons, it is better to automate test cases that will be reused in later software testing phases [54].

This study presents an overview of the main software automated testing framework. From out the review, we emphasize that using the programmable automation testing approach is preferable. However, due to its high cost, new techniques should be developed to overcome this problem.

6. REFERENCES

[1] Q. A. Malik, "Combining Model-Based Testing, Stepwise Formal Development," Abo Akademi University, Department of Information Technologies Joukahaisenkatu, [PhD Thesis], 2010.

[2] Banerjee, B. Nguyen, V. Garousi and A. Memona, "Graphical User Interface (GUI) Testing Systematic Mapping and Repository," Information and Software Technology, vol. 55, no. 10, p. 1679–1694, 2013.

[3] P. Yadav and A. Kumar, "An Automation Testing Tool Using Selenium," International Journal of Emerging Trends & Technology in Computer Science (IJETTCS), vol. 4, no. 5, pp. 068-071, 2015.

[4] G. Tassey, "The Economic Impacts of Inadequate Infrastructure for Software Testing," National Institute of Standards and Technology Acquisition and Assistance Division, 2002.

[5] Jain and S. Sharma, "An Efficient Keyword Driven Test Automation Framework for Web Applications," International Journal of Engineering Science & Advanced Technology, vol. 2, no. 3, pp. 600-604, 2012.

[6] K. M. Mustafa, R. E. Al-Qutaish and M. I. Muhairat, "Classification of Software Testing Tools Based on the Software Testing Methods," Second International Conference on Computer and Electrical Engineering, vol. 2, 2009.

[7] O. A. Lemosa, F. C. Ferrari, M. M. Eler, C. J. Maldonado and P. C. Masiero, "Evaluation Studies of Software Testing Research in Brazil, In The World: A Survey of Two Premier Software Engineering Conferences," Journal of Systems and Software, vol. 86, no. 4, p. 2013, 951–969.

[8] Santiago, W. P. Silva and N. L. Vijaykumar, "Shortening Test Case Execution Time for Embedded Software," Second International Conference on Secure System Integration and Reliability Improvement, 2008.

[9] R. K. Chauhan and I. Sing, "Latest Research and Development on Software Testing Techniques and Tools," International Journal of Current Engineering, Technology, vol. 4, no. 4, 2014.

[10] Singh and B. Tarika, "Comparative Analysis of Open Source Automated Software Testing Tools: Selenium, Sikuli, Watir," International Journal of Information and Computation Technology, vol. 4, pp. 1507-1518, 2015.

[11] Divya and S. D. Mahalakshmi, "An Efficient Framework for Unified Automation Testing: A Case Study on Software Industry," International Journal of Advanced Research in Computer Science & Technology, vol. 2, 2014.

[12] T. Kanstrén, "A Review of Domain-Specific Modelling, Software Testing," The Eighth International Multi-Conference on Computing in the Global Information Technology, 2013.

[13] Pillai, "Designing Keyword Driven Framework mapped at Operation Level," 2017. [Online]. Available: http://www.automationrepository.com/2012/08/keyword-driven-framework-mapped-at-operation-level-part-1/.

[14] S. Thummalapenta, S. Sinha, N. Singhania and S. Chandra, "Automating Test Automation," 34th International Conference on Software Engineering (ICSE), 2012.

[15] V. N. Maurya and R. Kumar, "Analytical Study on Manual vs. Automated Testing Using with Simplistic Cost Model," International Journal of Electronics and Electrical Engineering, vol. 2, no. 1, 2012.

[16] Jain, M. Jain and S. Dhankar, "A Comparison of RANOREX, QTP Automated Testing Tools, their impact on Software Testing," International Journal of Engineering, Management & Sciences (IJEMS) ISSN-2348 –3733, vol. 1, no. 1, 2014.

[17] J. Mishra, I. Ali and A. K. Upadhyay, "Automated Model Based Testing," International Journal of Engineering Research & Technology (IJERT), vol. 1, no. 4, 2012.

[18] V. Sangave and V. Nandedkar, "Generic Test Automation," International Journal of Science and Research (IJSR), vol. 4, no. 7, 2015.

[19] M. Sadiq and F. Firoze, "A Method for the Selection of Software Testing Automation Framework using Analytic Hierarchy Process," International Journal of Computer Applications, 2014.

[20] S. Maheshwari and D. C. Jain, "A Comparative Analysis of Different Types of Models in Software Development Life Cycle," International Journal of Advanced Research in Computer Science and Software Engineering, vol. 2, no. 5, 2012.

[21] J. A. Clark, H. Danb and R. M. Hierons, "Semantic Mutation Testing," Third International Conference on Software Testing, Verification, and Validation Workshops, vol. 78, no. 4, p. 345–363, 2011.

[22] O.-P. Puolitaival, T. Kanstrén, V.-M. Rytky and A. Saarela, "Utilizing Domain-Specific Modelling for Software Testing," The Third International Conference on Advances in System Testing and Validation Lifecycle, pp. 115-150, 2011.

[23] T. Kosa, M. Mernikb and T. Kosarb, "Test Automation of a Measurement System Using a Domain-Specific Modelling Language," Journal of Systems and Software, vol. 111, p. 74–88, 2016.

[24] S. Paydar and M. Kahani, "An Agent-Based Framework for Automated Testing of Web-Based Systems," Journal of Software Engineering and Applications, vol. 4, pp. 86-94, 2011.

[25] Ema and E. M. Reddyb, "Software Test Automation: An algorithm for solving system management automation problems," International Conference on Information, Communication Technologies (ICICT), vol. 46, pp. 949-956, 2015.

[26] T. R. Devi, "Propose Automated Software Testing Tools to Test Given Application Report Bugs," International Journal of Engineering Research and Technology (IJERT), vol. 2, no. 1, 2013.

[27] B. Bhondokar, P. Ranawade, S. Jadhav and M. Vibhute, "Hybrid Test Automation Framework for Web Application," International Journal of Engineering Research and Technology (IJERT), vol. 4, no. 4, pp. 1007-1012, 2015.

[28] M. Hammoudi, G. Rothermel and P. Tonella, "Why do Record/Replay Tests of Web Applications Break?," IEEE International Conference on Software Testing, Verification and Validation (ICST), 2016.

[29] B. N. Nguyen, B. Robbins, I. Banerjee and A. Memon, "GUITAR: an Innovative Tool for Automated Testing of GUI-driven Software," Automated Software Engineering, vol. 21, no. 1, p. 65–105, 2014.

[30] J. Tang, X. Cao and A. Ma, "Towards Adaptive Framework of Keyword Driven Automation Testing," IEEE International Conference on Automation and Logistics, 2008.

[31] Peethambaran, "Automated Functional Testing Using Keyword-driven Framework," Helsinki Metropolia University of Applied Sciences, Master of Engineering [Master Thesis], 2015.

[32] G. D. Lucca, A. Fasolino and F. Faralli, "Testing web applications," International Conference on Software Maintenance, p. 310, 2002.

[33] R. D. Craig and S. P. Jaskiel, Systematic Software Testing, Artech House Publishers, 2002.

[34] T. Pajunen, T. Takala and M. Katara, "Model-Based Testing with a General Purpose Keyword-Driven Test Automation Framework," Fourth International Conference on Software Testing, Verification and Validation Workshops, 2011.

[35] M. Leotta, D. Clerissi, F. Ricca and P. Tonella, "Capture-Replay vs. Programmable Web Testing," 20th Working Conference on Reverse Engineering (WCRE), 2013.

[36] M. Leotta, D. Clerissi, F. Ricca and P. Tonella, "Visual vs. DOM-Based Web Locators: An Empirical Study," International Conference on Web Engineering (ICWE), p. 322–340, 2014.

[37] H. Liu and H. Lieberman, "Programmatic Semantics for Natural Language Interfaces," Proceeding Extended Abstracts on Human Factors in Computing Systems, pp. 1597-1600, 2005.

[38] C. Kelleher and R. Pausch, "Lowering the Barriers to Programming: A Taxonomy of Programming Environments,Llanguages for Novice Programmers," Journal ACM Computing Surveys (CSUR), vol. 37, no. 2, p. 133–137, 2005.

[39] S. Srivastava, S. Gulwani and J. S. Foster, "From Program Verification to Program Synthesis," Proceedings of the 37th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages, vol. 45, no. 1, pp. 313-326, 2010.

[40] G. Little and R. C. Miller, "Translating Keyword Commands into Executable Code," Proceeding UIST '06 Proceedings of the 19th annual ACM symposium on User interface software and technology, pp. 135-144, 2006.

[41] F. Wang and W. Du, "A Test Automation Framework Based on WEB," IEEE/ACIS 11th International

Conference on Computer and Information Science, pp. 683-687, 2012.

[42] J. M, S. P and S. Prabu, "Web-Based Automation Testing Framework," International Journal of Computer Applications, vol. 45, no. 16, 2012.

[43] K. V. Arya and H. Verma, "Keyword Driven Automated Testing Framework for Web Application," 9th International Conference on Industrial and Information Systems (ICIIS), 2014.

[44] S. Singla and H. Kaur, "Selenium Keyword Driven Automation Testing Framework," International Journal of Advanced Research in Computer Science, Software Engineering, vol. 4, no. 6, pp. 125-129, 2014.

[45] K. Jha, "Development of Test Automation Framework for Testing Avionics Systems," 29th Digital Avionics Systems Conference (DASC), 2010.

[46] M. Yalla and M. Shanbhag, "Building Automation Framework Around Open Source Technologies," Proceeding of Software Testing Conference, pp. 6-9, 2009.

[47] S. Stresnjak and Z. Hocenski, "Usage of Robot Framework in Automation of Functional Test Regression," The Sixth International Conference on Software Engineering Advances (ICSEA), pp. 30-34, 2011.

[48] Madhavan, "Semi Automated User Acceptance Testing using NLP," Lowa State University, [Master Thesis], 2014.

[49] Cervantes, "Exploring the Use of a Test Automation Framework," IEEE Aerospace conference, 2009.

[50] E. Engström, P. Runeson and M. Skoglund, "A Systematic Review on Regression Test Selection Techniques," Information and Software Technology, vol. 52, p. 14–30, 2010.

[51] Swarnendu and R. Mall, "Regression Test Selection Techniques A Survey," An international Journal of Computing and Informatics, vol. 35, p. 289–321, 2011.

[52] Zarrad, "A Systematic Review on Regression Testing for Web-Based Applications," Journal of Software, vol. 8, pp. 971-990, 2015.

[53] P. K. Chittimalli and M. J. Harrold, "Recomputing Coverage Information to Assist Regression Testing," IEEE Transactions on Software Engineering, vol. 35, no. 4, p. 452–469, 2009.

[54] K. Dobolyi, E. Soechting and W. Weimer, "Automating regression testing using web-based application similarities," International Journal on Software Tools for Technology Transfer, vol. 13, no. 2, p. 111–129, 2011.

[55]