NOVATEUR PUBLICATIONS
INTERNATIONAL JOURNAL OF INNOVATIONS IN ENGINEERING RESEARCH AND TECHNOLOGY
[IJIERT] ISSN: 2394-3696 Website: ijiert.org
VOLUME 5, ISSUE 2, Feb. -2018

AUTOMATED TESTING FRAMEWORKS: ENSURING SOFTWARE QUALITY AND REDUCING MANUAL TESTING EFFORTS

Swamy Prasadarao Velaga
Sr. Programmer Analyst, Department of Information Technology

ABSTRACT

The purpose of this research study is to review and analyze the automated testing frameworks thoroughly with the main areas of focus on the quality assurance of the software and the minimization of testing that is done manually. The research aims and objectives are as follows: This is in relation to the historical development of these frameworks, their types, and their essential characteristics The examination of their effectiveness in enhancing software quality Determine the extent of diminished manual testing efforts The examination of the various ineffectiveness and drawbacks related to the frameworks [1]. Moreover, it presents an aim to reveal best practices, to assess their usage in CI/CD pipelines, and compare the cost and benefits impatient of the practices; to analyze their compliance with the current legislation and industrial standards and to investigate the trends in the further development of the field. Automated testing frameworks (ATFs) are becoming increasingly popular and are practically valuable in custom software development[1]. ATFs are utilized to reduce the manual effort to perform testing and ensure high-quality, robust software. ATFs can be easily implemented for integration and system testing, but there are challenges to implement unit testing automation.Software testers use a number of testing techniques to assess the quality of software, which helps in ensuring that the software meets its requirements and is delivered with minimal defects. Manual testing is performed by a human executing an application to verify different operations of an application and to validate different test conditions. Although manual testing will always have a role in software testing, it does have some limitations. Some of the limitations of manual testing can be overcome by using test automation[1]. Test automation is the use of software to control the execution of tests and to compare the actual results with the expected results. Test automation can be used in repetitive tasks, tasks that are difficult or impossible to do manually, and tasks that are performed under extreme conditions. Test automation can perform testing effectively and efficiently. Automated Testing Frameworks (ATFs), also known as test automation frameworks, are an approach to automate software testing activities to verify different test scenarios and conditions of a software system. The ATF combines automated test scripts and test data and repositories with components that execute pre-test and post-test actions, and reporting to form a structure that supports automated testing[2].With the growing acceptance of agile development and DevOps methodologies, the time allotted for testing has been dramatically reduced. As a result, testers are under pressure to quickly validate the quality of the software and are turning to automated testing. Many test automation frameworks have been developed to assist testers with executing and reporting results of their automated tests. These frameworks can also help reduce the effort involved in test script development.

Keywords: Automated Testing, Testing Frameworks, Software Quality, Manual Testing, Software Testing, Test Automation, Continuous Integration, Continuous Deployment, Testing Strategies, Software Development, Testing Efficiency Testing Tools, Quality Assurance, Testing Challenges, CI/CD Pipelines.

INTRODUCTION

Automated testing frameworks are crucial in software testing as they help in minimizing the proportion of the manual testing efforts hence enhancing testing efficiency and thus decreasing the overall testing costs. Also,

this leads to the shortening of the testing cycle that are in the process. Moreover, these frameworks do enable the generation of test scripts, which consequently augments the level of test coverage. Manually written test scripts without an automation framework are flawed and could affect the overall test efficiency [3]. Testing frameworks for generating test scripts also mean that testing can occur at more regular intervals during the SDC life-cycle, meaning that errors are caught early and more quickly. The probability of defects is distributed over the Software Development Life Cycle so the earlier they are found, the cheaper they are to correct. In summary, the automated testing frameworks hold a central function in increasing the overall competence of software testing. Automated testing frameworks can also aid in increasing the quality of the results through the level of accuracy and precision, which proves the functioning of the software. Through decreasing the usage of manual testing, these frameworks remove human influence and give steady, repeatable tests. This eventually enhances the credibility of the testing process together with the quality of the developed software [4]. Also, it can be connected to the CI/CD system, which enables automated testing and subsequent integration of new features and changes into the application. This integration also enhances the development process because problems that arise are quickly pin-pointed and resolved [5]. Yet another important advantage of the automated testing frameworks is that they are easily portable and grow in parallel with the growth of the software project. These frameworks become beneficial as the code base grows, and the overall software becomes more sophisticated; they are capable of managing more tests and offering test coverage. This scalability is very important, in order to ensure that the software is still efficient as it grows and changes with time [5].

ATFs can be utilized by testing teams without specialized programming knowledge. They are much easier to implement when compared to lower level testing solutions. Test harnesses, which are used to test functions and libraries in isolation from the user interface, are examples of lower level test solutions that need to be manually developed. Even though ATFs are easier to implement, they do have drawbacks. ATFs generally only test the external attributes of the application [5]. They do very limited testing of the internal code structure, design, or program logic. Consequently, they may not be able to catch certain categories of bugs. Typically, ATFs are most effective in testing applications with simple flow control and limited state transitions. High level testing tools or ATFs may not be capable of handling dynamic conditions that require variable data or timing control. In summary, the ease of implementation of ATFs is achieved at the cost of limited testing capabilities. Consequently, ATFs may substantially reduce, but not entirely replace, manual testing efforts. It ensures that the software performs its operations accurately and reliably according to the user's expectations. Quality assurance through testing is therefore a necessary activity in software development to assess and control the quality of software products [6]. However, testing is a time-consuming and labor-intensive task in the software development life cycle. While manual testing is important to evaluate real customer usage and experience, it is usually performed for specific test scenarios and with limited test coverage. As a result, manual testing efforts are costly and ineffective, often leading to a compromise between test coverage, time, and budget.

RESEARCH PROBLEM

The main research problem in this study is to explore how one can ensure software quality with high efficiency through the employment of automated testing frameworks in reducing the amount of efforts devoted to manual testing. This includes deriving essential characteristics of successful frameworks, measuring the effects of the formulated components on quality of software, assessing the opportunities and impediments relating to implementation of such frameworks, as well as outlining the superior patterns and potential developments in

the area. Furthermore, the evaluation of the integration of these frameworks into CI/CD processes, along with the cost savings consideration and compliance with relevant rules and standards are also going to be studied in the current research. Automated testing is a priority for the testing community. This was highlighted as a research priority from the Software Engineering Testing Summit [7]. As software continues to increase in complexity, the importance of testing as a mechanism for ensuring software quality also increases. It is of vital importance that an appropriate testing framework is utilized alongside the development process. Sun Microsystems reports that "half of software production costs are eaten up by testing, yet with inadequate results". Furthermore, as Sun Microsystems points out, this "expense only seems to be growing". The use of a testing framework to automate the testing processes can reduce manual efforts. Over the past few years, the software testing community has produced a large number of disparate tools that focus on various aspects of the testing process to "automate" testing. For example, tools exist that automate test case generation, user-interface or GUI testing, regression testing, and load testing. Unfortunately, these tools are not often integrated [7].

LITERATURE REVIEW

A. EVOLUTION OF AUTOMATED TESTING

Automated testing has become a popular and effective way to test software. It is well-known for its benefits, including savings in test execution time, reduction of human errors, increased test coverage, and better bug detection. However, the establishment of automated testing is not simple. Although it can be difficult and time-consuming to implement, the overall effort to perform this type of testing is still much less than performing manual testing every time a new version of the software is released [7]. As a result, more and more test teams are moving towards automated testing. This chapter describes the history of automated testing along with its precursor, manual testing. Information about the first test script developed lays the foundation for discussion on the key development steps of the Automated Testing Framework (ATF). Supporting test software is critical for the success of automated testing, and it involves the ATF that encompasses methods and guidelines. The ATF can be implemented in an organization using certain popular test development tools. These test tools, when used, help the test team rapidly develop, deploy, and stabilize automated tests and ensure their maintenance over time. The growth of application development has increased the necessity of its testing phase to ensure its quality and proper functionality. With the increase in manual testing efforts, automated testing has come to the rescue. Automated testing executes the same process as manual testing but with the help of automation tools, scripts, and software. These tools significantly reduce manual efforts, time, and cost [7,8].

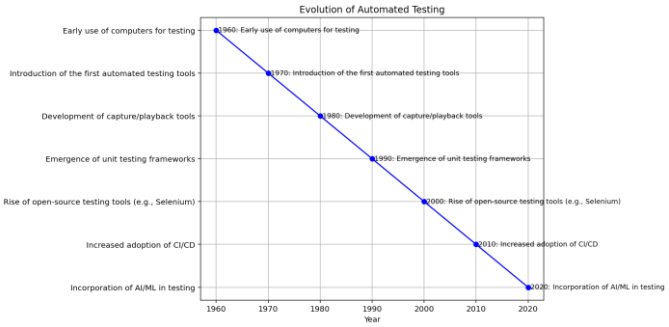

Fig. 1 Evolution of Automated Testing

The early days of software testing involved large teams of testers working long hours to test software to identify and correct errors. This was necessary because practically no software testing was automated in the

early days of computing. With the advent of the first commercial mainframes in the 1950s, pioneers in the testing field began to write software to test the reliability of code. As programming advanced, software testing also advanced to include areas such as functional and performance testing. In 1979, Professor Glen Myers introduced the concept of automatic test case generation and this became a foundation for the generation of many commercial testing tools [9]. The 1980s saw a surge in the development of tools to support software testing. Tools were developed to support unit, functional, and performance testing as well as test data generation and test management. Over the next three decades, many new tools were developed along with enhancements to existing tools as software testing evolved to keep pace with rapidly advancing software development practices.

B. TYPES OF AUTOMATED TESTING FRAMEWORKS

In a data-driven testing framework, test input or output values are retrieved from data files rather than being hardcoded in the script. These data files can come in the form of Excel files, XML files, keyword files, database files, etc. There are different types of data-driven frameworks such as key-driven, table-driven, and the standard data-driven framework. Three types of automated testing frameworks can be discerned at a high level: 1. Those utilizing GUI based scripting/programming capabilities of general automated testing tools. For example, keyword-driven, data-driven and hybrid-driven testing are scripts developed using testing tools that are capable of recording/playback of testing activities. 2. Domain-specific or specialized testing frameworks. These are built using commercial or open-source tools, with scripting/programming capabilities like those in type 1. However, additional tool-specific modules may be included based on the identified domain. 3. Development of custom, specialized testing frameworks, utilizing GUI and non-GUI testing capabilities of general automated testing tools and/or specifically developed modules [10].

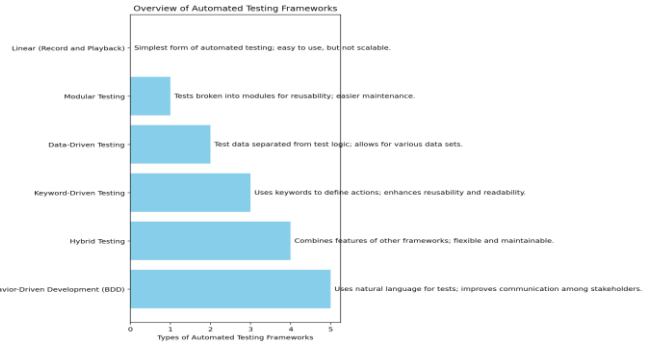

Fig. 2 Types of Automated Testing Frameworks

Since general capabilities would comprise GUI testing, test tools would have capabilities like statically mapping software object properties with corresponding test data and results, and programmable testing logic based on mapped object properties. Programming is required in order to implement custom testing logic and patterns. Automated testing frameworks provide standardized modules and logic for specific test scenarios. Lack of domain support out-of-the-box in general automated testing tools requires manual coding for a major part of testing logic. Custom automated testing codes/modules segmented based on specific areas in a software package may be used together with openly available general testing tools. Such segmented and general fragmented test codes become easily maintainable - independent of unstable test conditions and easily modifiable under changing test requirements [11].

C. KEY FEATURES AND COMPONENTS OF EFFECTIVE TESTING FRAMEWORKS

The ability to easily execute tests in a framework is a key feature. Other features such as test isolation, test, and stub code generation can easily ensure that this is always possible. Simple test execution is always a feature of a good testing framework. The inability to rapidly execute tests represents a testing bottleneck. Another desirable feature of test frameworks is the ability to provide meaningful analysis of test coverage. Management of test coverage analysis within the framework ensures that developers perform analysis at the appropriate intervals. If developers must rely on external tools to perform test coverage analysis, the analysis may only be performed intermittently [11]. Providing tools allowing developers to perform covered-by and implements/extends type queries helps ensure test coverage statements remain in synchronization with test code. Four features are central to the design of modern testing frameworks: the ability to easily construct and execute tests, test isolation, support for the creation of reusable test components, and test coverage analysis and management. Effective testing frameworks typically provide several components to help developers easily create and execute tests. Test runners automate the execution of tests, and test harnesses provide auxiliary mechanisms that help execute tests [12]. Examples of mechanisms provided by test harnesses include object creation and initialization, method invocation, event handling, and GUI manipulation. Test drivers are used to execute the code modules being tested, and stubs and mock objects are typically used inside the test harness to replace real objects and to simulate the behavior of the system or of system components.

D. IMPACT ON SOFTWARE QUALITY

The impact of automated testing on software quality is not just because of the characteristics of automated testing (e.g., repeatability, reliability), but more because automated testing can leverage advanced testing techniques to address more aspects of the software under test. Most automated testing frameworks use manual test cases as their input, but the testing techniques used in manual testing can be extended and applied in automated testing. For example, exploratory testing is a manual testing technique that leverages the tester's knowledge and experience to perform testing and is one of the most effective and efficient testing techniques [12,13]. Due to time constraints, limited repeatability, and the lack of traceability, many argue that exploratory testing is not suitable for all testing activities, especially in larger software projects. However, using an exploratory testing approach within an automated testing framework can address these issues and make exploratory testing feasible for larger projects. The goal of any software testing effort is to ensure the quality of an application before it is released to the users. Although testing is vital in this process, it is considered labor-intensive work that consumes large amounts of effort. Testing consumes the most effort (and therefore money) relative to all other development activities. Performing manual tests is one of the most time-consuming and expensive components of the testing process. For each software release, the user interface must be manually tested using a variety of test cases. Documenting and executing these test cases demands considerable effort. Automated testing can assist with the reduction of manual testing efforts [14]

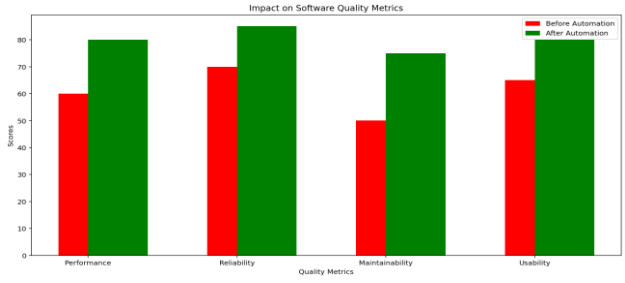

Fig. 3 Impact on Software Quality Metrics

E. CI/CD PIPELINES

Including automated tests in the CI/CD pipeline ensures that the latest software changes pass all high-priority tests before releasing the software. The pipeline can be configured to run all test suites or specific test cases at different stages of the pipeline for a selected target environment. The execution results will determine if the software is ready to progress to the next stage or if it should be rejected and the development team notified of the failure [14]. The CI/CD system interacts with the automated testing framework to run the identified tests and reports the test results. It will also inform the team which tests are flaky and have produced an inconclusive result, suggesting that the team rerun the tests.

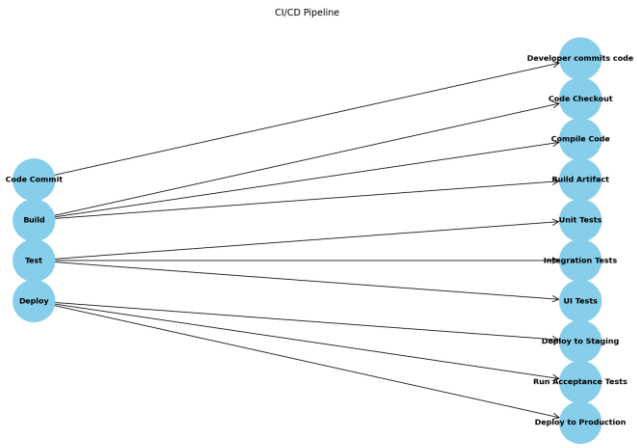

Fig. 4 CI/CD Pipeline

API-level testing has the advantage of faster execution than GUI testing, but it requires an understanding of the test-to-test dependencies related to the test data setup and test data assertion for test execution. API testing is more reliable than GUI-based testing due to the removal of test data setup dependencies related to the test environment configuration and the test data assertion dependencies related to the test results. For software projects with a testing pyramid approach that emphasizes unit testing, the project requires a high standard of unit test coverage with a focus on critical business logic components [15,16]. The unit tests should include individual tests and tests that mock the dependencies.

CONTRIBUTIONS

My contribution in this study is to outline how automated testing shows its power by reducing the time to perform tests. Once an automated test is created, it can be run multiple times without executing any of the test steps required for manual retesting. Automated tests are executed by a computer, allowing software teams to run tests independent of any human interaction. The ability to run automated tests in parallel to manual testing efforts helps reduce the duration of the testing process. Efforts required to test a software application are greatly reduced by the ability to reuse test scripts. When a new software release has been developed or changes have been made to an existing application, manual testers typically use the same set of test cases and retest the software. If manual testers find the expected results during test execution, they typically need to perform the same tests over multiple test cycles, each time new builds of the software are deployed. By executing automated tests, multiple configurations of the build could be tested simultaneously. If the automated tests pass and the primary configurations are covered, manual testers can focus their effort on more exploratory testing. In this framework, test drivers play a key role and are reusable objects. When a specific test driver is

designed and developed, then that driver can work with various test components. Then, test cases can be derived automatically from those test cases. The designers can focus on deriving new specialized test cases for the specific components. Existing test case derivation tools can be utilized for this task. Test cases can be validated through formal methods before the derived test cases are submitted to the framework for execution. The framework also supports distributed test execution for specific features or for feature combinations.

SIGNIFICANCE AND BENEFITS

Automated testing frameworks, in conjunction with manual testing efforts, are likely to increase the coverage of testing of various software modules, services, or applications and thereby increase the overall quality of these software. Despite such advantages, companies may be unwilling to invest in automated testing frameworks due to the additional cost in training their staff to use these frameworks [17]. The existing cost benefits of utilizing these automated testing frameworks have been largely empirical in nature or very specific to certain types of software. Freeware and open-source automated testing frameworks can help reduce costs associated with the testing of software. They can provide significant reduction in time and cost required for developing and executing custom testing programs and scripts. These testing frameworks typically have well-defined testing components that can be easily augmented, adapted, or modified to fit testing of specific software components or subsystems [18]. Many companies, small and large, make heavy investments in the development of their software products. Business applications, as well as specialized software for industries such as aerospace, banking, and healthcare, are developed with great effort and expense. A company's investment in software development is often only one part of the cost. Equally large costs can be incurred in software maintenance, including activities such as software testing - to ensure that the software works correctly and as expected - and software debugging - to locate and correct errors. Reduction of testing effort by benefiting from automated test generation and execution at well-defined project points will be one way to use these models. Other factors also play an important role in determining the effectiveness of an automated testing framework [19]. The level of automation, manual testing override, and code stability interact with the framework and have their own specific considerations.

CONCLUSION

This paper focused on reviewing and presenting a high-level overview of some emerging automated testing frameworks that can be used to ensure software quality and reduce manual testing efforts. These automated testing frameworks have the potential to cope with the rapidly increasing complexity of software systems. We identified some current challenges and future research directions of these automated testing frameworks. These challenges and future research directions need to be addressed so that using these automated testing frameworks becomes accessible to quality assurance engineers and the maintained source code of the tested software systems can be enhanced. Ensuring software quality is the main goal of software testing, but it is time consuming and effortful when done manually. With the increasing complexity of software systems, most software development companies are still at a crossroad whether to increase their manual testing efforts or switch to automated testing to cope with the increasing complexity of software systems to ensure software quality. The research community has been addressing this issue, but the available solutions are not practical for industry usage. In this paper, we present a high-level overview of some emerging automated testing frameworks that can be used to ensure software quality and reduce manual testing efforts. These automated testing frameworks have the potential to cope with the rapidly increasing complexity of software systems. As the size and complexity of software increase, the software industry is facing increasing pressure to ship high-

quality software within short development cycles. The quality of software is considered one of the most important factors that determine the success of software. It is also considered a determining factor for the market competitiveness of software. However, testing software is labor-intensive and time-consuming, especially with the increase in different types of software such as web, mobile, and cloud-based software. The manual testing effort required to test such software is huge, especially in regression testing, performance testing, and security testing of software. Automated testing is the solution to such manual testing effort and is widely used in the industry despite some of the challenges associated with automated testing. Automated Testing Frameworks: Ensuring software quality and reducing manual testing efforts

REFERENCES

[1] E. Dustin, T. Garrett, and B. Gauf, Implementing Automated Software Testing. Pearson Education, 2009.
[2] J. Watkins, Testing IT : an off-the-shelf software testing process handbook. New York: Cambridge University Press, 2001.
[3] E. Dustin, J. Rashka, and J. Paul, Automated Software Testing. Addison-Wesley Professional, 1999.
[4] [W. E. Lewis, Software Testing and Continuous Quality Improvement, Third Edition. CRC Press, 2016.
[5] M. Fewster and D. Graham, Software test automation : effective use of test execution tools. Reading, Ma: Addison-Wesley, 1999.
[6] D. Graham, M. Fewster, L. Copeland, and A. Wesley, Experiences of test automation : case studies of software test automation. Upper Saddle River Etc.: Addison-Wesley, Cop, 2012.
[7] G. Alpaev, Software testing automation tips : 50 things automation engineers should know. [Berkeley, CA]: Apress, 2017.
[8] K. Li and M. Wu, Effective Software Test Automation. John Wiley & Sons, 2006.
[9] D. J. Mosley and B. A. Posey, Just enough software test automation. Upper Saddle River, Nj: Yourdon Press, 2002.
[10] S. Saeed, Knowledge-Based Processes in Software Development. IGI Global, 2013.
[11] A. Bhargava, Designing and implementing test automation frameworks with qtp. Packt Publishing Limited, 2013.
[12] C. Kaner, J. Bach, and B. Pettichord, Lessons Learned in Software Testing. John Wiley & Sons, 2011.
[13] L. G. Hayes, The automated testing handbook. Richardson, Tx: Software Testing Institute, 2004.
[14] S. Nakov, Fundamentals of computer programming with C# : The Bulgarian C# programming book. Sofia, Bulgari: Svetlin Nakov & Co, 2013.
[15] B. Homs, Fundamentals of Software Testing. London: Wiley, 2013.
[16] J. A. Whittaker, J. Arbon, and J. Carollo, How Google Tests Software. Addison-Wesley, 2012.
[17] H. Nguyen, M. Hackett, and B. K. Whitlock, Happy About Global Software Test Automation. Happy About, 2006.
[18] L. Ryan, Happy about online networking : the virtual-ly simple way to build professional relationships. Cupertino, Calif.: Happy About.Info, 2006.
[19] G. Paskal, Test Automation in the Real World. 2017.