Scientific Publications

Journal of Artificial Intelligence and Big Data, 2022, 2, 1351
www.scipublications.com/journal/index.php/jaibd
DOI: 10.31586/jaibd.2022.1351

Article

Survey of Automated Testing Frameworks and Tools for Software Quality Assurance: Challenges and Best Practices

Vaibhav Maniar [1,*], Aniruddha Arjun Singh Singh [2], Rami Reddy Kothamaram [3], Dinesh Rajendran [4], Venkata Deepak Namburi [5], Vetrivelan Tamilmani [6]

[1] MBA / Product Management, Oklahoma City University, USA
[2] ADP, Sr. Implementation Project Manager, USA
[3] MS in Computer Information Systems, California University of Management and Science, USA
[4] MSC. Software Engineering, Coimbatore Institute of Technology, USA
[5] Department of Computer Science, University of Central Missouri, USA
[6] Principal Service Architect, SAP America, USA

*Correspondence: Vaibhav Maniar (vaibhav.maniar@gmail.com)

Abstract: Automated testing and software quality assurance (SQA) practices are essential for ensuring the reliability, scalability, and maintainability of modern software systems. This paper presents a review of widely used automated testing frameworks, including Driven, Data-Driven, Behavior-Driven Development (BDD), and Record/Playback approaches, outlining their methodologies, benefits, and limitations in different development contexts. In parallel, it examines established SQA techniques such as Test-Driven Development, static analysis, and white-box testing, which provide systematic methods for defect detection and quality improvement. The study further examines the role of practical tools, such as Selenium, TestNG, and JUnit, in supporting test automation and validation activities. In addition to highlighting technical capabilities, the paper identifies common challenges faced in automation, including incomplete requirements, integration complexities, and maintaining evolving test suites. Recommended best practices are provided to address these issues, offering guidance for organizations seeking to strengthen their software testing processes through structured frameworks, adaptive techniques, and reliable automation tools.

Keywords: Automated Testing, Software Quality Assurance, Testing Frameworks, Test Automation Tools, Best Practices

How to cite this paper:
Maniar, V., Singh, A. A., Kothamaram, R. R., Rajendran, D., Namburi, V. D., & Tamilmani, V. (2022). Survey of Automated Testing Frameworks and Tools for Software Quality Assurance: Challenges and Best Practices. *Journal of Artificial Intelligence and Big Data, 2*(1), 176–186. DOI: 10.31586/jaibd.2022.1351

Received: September 8, 2022
Revised: October 31, 2022
Accepted: November 30, 2022
Published: December 27, 2022

1. Introduction

Software has become an important part of daily lives because it affects so many people in so many areas of their lives. Therefore, we need safe and trustworthy software. Programming mistakes are unavoidable due to the inherent fallibility of people and the inherent human element in software development. Some software mistakes, known as bugs, can have fatal consequences for the live operation. Fixing these mistakes later on in the development process cost more than fixing them earlier [1]. Consequently, software testing is the last step in the process of ensuring a product is right and is an essential part of software quality assurance. Improving the product's quality always boosts client confidence, which in turn promotes the business's economics [2]. What this means is that a high-quality product has no flaws because of the thorough testing that went into making it. Testing is running a software to identify and fix any bugs.

The software business faces a significant difficulty with the cost of testing, which is often stated by both academics and industry to be over 50% of the entire development cost and seldom less than 20% Moreover, the software business is undergoing a transformation towards a more nimble and responsive setting, prioritizing the continuous integration, development, and deployment of products to clients [3]. Speedier and more frequent input on software quality is necessary in this setting, which therefore places new demands on testing. Commonly, this is cited to support the idea of increasing test automation.

An automated software test utilizes a scripting language or other external automation tools to replicate the procedures of a manual test case. Automation testing requires a well-defined manual testing procedure to automate tests, as this involves replacing the current manual process with an automated one [4]. Automating the execution phase is the main focus. Because it may be reused without further work, it speeds up the execution of tests. It definitely takes a long time to accomplish this on the first run. But once the scripts are prepared, the human tester can automate their execution on the SUT. It significantly reduces expenses during software testing. Software testers can automate software testing with the help of a test automation framework, which is a collection of preset ideas, concepts, assumptions, and implementations. It is also possible to think of it as a collection of encapsulated functions that make automation easier. In addition to improving the reusability of automated tests, these studies significantly aid in managing and tracking the execution of business test scenarios [5].

Organization of the Paper

The paper is organized as follows: Section II outlines automated testing frameworks, Section III discusses supporting techniques and tools, Section IV highlights challenges and best practices, Section V reviews related literature, and Section VI concludes with future work.

2. Automated Testing Frameworks

Automated testing frameworks provide structured approaches to design, implement, and execute tests efficiently. Common types include Keyword-Driven, Data-Driven, Behavior-Driven (BDD), and Playback/Record frameworks. Each framework offers unique methods for organizing test scripts, separating test logic from data, or enabling collaboration between development and testing teams, helping improve test coverage, maintainability, and execution efficiency. Table 1 shows the Automated Testing Frameworks are shown in below:

Table 1. Comparative Analysis of Automated Testing Frameworks

Framework	Key Features	Advantages	Limitations	Best Suited For
Keyword-Driven Testing	Uses keywords stored in external files; separates test logic from scripts.	High reusability, easy maintenance, reduces programming effort by defining user keywords.	Initial setup effort is high; requires a well-defined keyword library.	Applications requiring frequent test modifications.
Data-Driven Testing	Separates test data from logic; executes single test scripts with multiple data sets.	Reduces the number of scripts; high reusability; flexible and scalable.	Debugging can be more challenging if test data is large, as it requires a careful design of the test data matrix.	Applications with large input variations.
Behavior-Driven Testing (BDD)	Extends TDD; focuses on customer-centric acceptance tests; uses natural language scenarios.	Improves communication between developers, testers, and stakeholders, while enhancing code quality.	Requires cultural shift and training; higher initial effort.	Agile projects and systems require stakeholder collaboration.

Playback/Record Testing	Creates test scripts by recording user interactions; minimal coding effort.	Very easy to implement; quick automation setup.	Poor maintainability; unstable with frequent UI changes.	Small projects or as an entry-level automation approach.

2.1. Keyword-Driven Testing Framework

One application-agnostic methodology is keyword-driven testing, which explains what needs to be done on the application in question using data tables and terms that speak for themselves [6]. The external input data file stores not only the test data but also the instructions that are part of the test scripts. Words like "keyword" describe these instructions. There are two types of keywords: base keywords and user keywords. Base keywords are those that are built into the libraries. Combinations of basic keywords and other user keywords are what define user keywords in the test data. Reduced programming effort and base keyword requirements result from the ability to generate new user keywords from test results. Adding, removing, and modifying the test scripts is possible. Figure 1 shows how modifying the test script can help with parameterization and breaking a test into different activities.

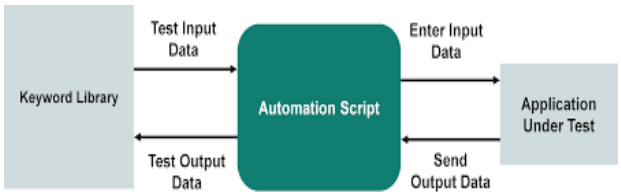

Figure 1. Keyboard Driven Testing Framework

- **Flow of Keyword Driven Framework:** The automated testing procedure begins with the test-driven script contacting the main function library, which houses the code for keyword identification. For every keyword, the function library locates the appropriate function. Next, the function is executed, which runs the test application through all of its required operations. Upon completion of the function, control is sent back to the main script, which then reads the next keyword. This procedure is repeated until all keywords have been analyzed and the functions linked to them have been run.

2.2. Data-Driven Testing Frameworks

This framework is an example of a data-driven testing strategy that aims to decouple test data and test logic. By making use of a structured test data matrix, it is possible to run a single test scenario with various data sets [7]. The framework can reduce the number of scripts required to cover all conceivable combinations of test cases by grouping tests in this way. This enables the efficient and versatile execution of a comprehensive suite of test cases with minimal coding effort, as changes to the test data do not require revisions to the underlying test scripts. This framework's primary value lies in the reusability it offers, which in turn reduces the overall number of scripts required to cover all potential permutations of test cases. This allows it to cover all test situations with minimal code. The framework's strength lies in its ability to isolate the test script code from changes made to the test data matrix. Various test data values can be run in a single scenario, as shown in Figure 2.

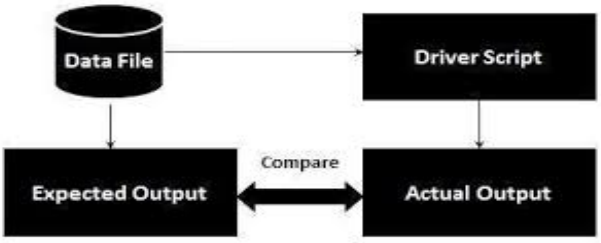

Figure 2. Data Driven Testing Framework

2.3. Behavior-Driven Testing Frameworks

A development methodology known as Behaviour-Driven Development (BDD) has emerged as an offshoot of Test-Driven Development (TDD) [8]. Improving code quality and minimizing error rates in software is where it's at. The goal is to build automated acceptance tests that accurately reflect complex system scenarios or stories. Building on top of TDD, BDD mandates that testers document the system's behaviour from the perspective of the client in acceptability tests. Figure 3 shows the main work produced by BDD testers, in contrast to classical unit tests that primarily evaluate the internal functionality of classes.

Figure 3. BDD Workflow

2.4. Playback Testing Framework

This automated testing framework aims to generate linear scripts using recording and playback apps, following the recording-playback concept. It is easy to implement the test with this framework, which is an advantage. Creating test scripts is as simple as recording. However, at the same time, there are several shortcomings with this framework; it cannot judge logic, resulting in a poor maintainability for the recording–playback automated testing framework. Moreover, it often leads to redundant or unstable scripts when applications undergo frequent changes, as shown in Figure 4. It is also less flexible for large-scale or complex testing needs. The recorded scripts usually require manual modifications to adapt to new scenarios. Therefore, this framework is considered more suitable for small projects or as an entry-level automation approach rather than for long-term enterprise use.

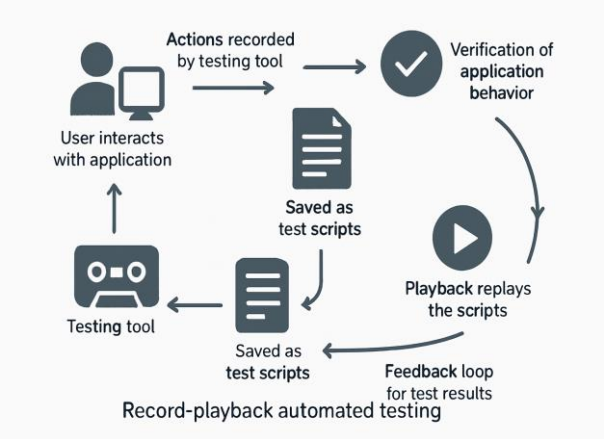

Figure 4. Record-Playback Testing Framework

3. Techniques and Tools Supporting Software Quality Assurance

This is essential to ensure that errors and defects are identified and eradicated before the application's implementation, as the development of all software systems should adhere to high-quality standards. Achieving this quality in software requires consistent use of dependable tools and defined methodologies throughout the development life cycle [9]. These techniques and tools provide systematic approaches for verification, validation, and continuous improvement, ensuring that software not only meets functional requirements but also maintains robustness, reliability, and long-term maintainability.

3.1. Techniques

Software Quality Assurance techniques are methods used to ensure software meets quality standards. They include static methods, such as code reviews and inspections, and dynamic methods, including unit, integration, and system testing. Practices like TDD and BDD help detect defects early and improve software reliability as shown in Figure 5.

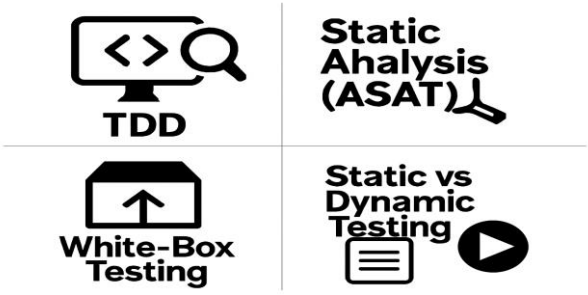

Figure 5. Techniques in Software Quality Assurance

- **Test-Driven Development:** In test-driven development, the goal is to create software by directing its design by means of automated tests, as mentioned before [10]. The conventional waterfall methodology involves creating test definitions separate from code, in accordance with the requirements. Automated testing doesn't necessarily require coding. The ratio of passing tests to total tests is a measure of product quality.
- **Static Analysis Tool:** Software systems can have their source or binary code scanned for a list of pre-defined faults by Automated Static Analysis Tools (ASATs). Both functional and maintainability issues, such as faulty logic or resource leaks, as well as non-compliance with best practices or style convention

breaches, can be detected by ASATs according to their configuration settings [11]. These days, ASATs are as essential to software quality assurance strategies as testing and manual code review. By paying attention to the alerts sent by ASATs, development teams can fix bugs before they make it into the final product.

- **White-Box Testing:** This method of testing requires the tester to have knowledge of the software's internal structure and implementation [12]. The selection of test cases in white-box testing is based on the implementation of the software entity. The tester's programming abilities and internal view of the system are utilized to construct the test cases. Full understanding of the source code is required for this type of testing. To verify that the software is functioning as intended, the tester chooses inputs to run it through its paces and reviews the results. White-box testing goes by a few different names: structural, transparent, glass, clear, logic-driven, and open-box.

3.2. Tools

Software Quality Assurance tools support the implementation of SQA techniques by automating testing, analysis, and reporting. Examples include Selenium, Junit and TestNG. These tools enhance efficiency, consistency, and accuracy in verifying software quality.

- **Selenium:** The Selenium web driver (Selenium 2.0), Selenium IDE (Selenium 1.0), and Selenium RC (Selenium 1.0) are all parts of the larger Selenium software automation framework. To create test scripts, one can use the Selenium IDE, an IDE [13]. Selenium test case recording, editing, and debugging is made possible with this Firefox add-on. In addition to creating the test scripts, it logs every action the user takes. For some time, Selenium's primary focus was on RC, or Remote Control. Using the JavaScript program known as Selenium Core causes Selenium RC to be slower than the Selenium WebDriver.
- **TestNG:** The testing framework TestNG was made to fix the problems with the JUnit testing framework. With its enhanced features, it surpasses JUnit in power. Many different types of tests, including unit, functional, and integration tests, are all supported by TestNG. Within the suggested architecture, TestNG is integrated with Eclipse to generate test reports and run multiple test cases simultaneously. All of the test cases, succeeded and failed, are detailed in these reports.
- **JUnit:** The JUnit framework's primary goal is to facilitate the creation, administration, and execution of unit tests for Java programs. The JUnit framework states that each unit test must evaluate the expected behaviour of a single class method. The primary features of JUnit include the following: the ability to define test cases or suites to be executed; the execution of a specific test case or suite; and finally, the collecting and presentation of test results.

4. Challenges and Recommended Best Practices

Software quality assurance faces challenges such as scalability, integration complexities, and domain-specific constraints. Recommended best practices include adopting flexible frameworks, automating test processes, ensuring comprehensive coverage, and continuously improving tools and methodologies to enhance reliability and maintainability.

- **Software Requirement:** Software quality is a major concern for clients. However, software requirement collecting is the root cause of quality suffering [14]. There is a potential for quality failure due to improper requirement

collecting, poor and incomplete requirements, inadequate time, and lack of focus. Gathering the correct requirements at the right time is the first hurdle.

- **Requirements Collection Period:** There consequences for the remainder of the period if requirements collecting is left unfinished. When requirements are not clearly defined, projects can fail. In the meantime, let's talk about how to avoid it during the requirements collection phase.
- **Avoid Defects:** Software development should not do something that is required by the product specification, yet it does [15]. This is an example of a flaw. A software flaw is anything that the product description says it shouldn't do. A software defect is any behavior that is not explicitly stated in the product specification. Imperfections in software are defined as features that should work but don't and aren't part of the product specification.
- **Software Variant:** Developers typically make an effort to accommodate a large variety of consumers by making their program compatible with multiple platforms (e.g., web and mobile). In addition, there are no limitations on software versions imposed by validation methodologies. As a result, customer efforts should be minimized while maximum integration to project success is maximized when using appropriate validation methods, which should synchronize the target application requirements. There more mistakes in requirements synchronization, less profit, and less reuse as the number of versions grows.
- **Effective Strategy:** Effective maintenance techniques for automated tests is another difficulty [16]. Software changes over time, and automated tests could become ineffective, causing problems like false positives or errors to go unnoticed. Creating methods and resources for upkeep and revision of automated testing.

5. Literature Review

This review highlights key advancements in automated testing frameworks and software quality assurance. Approaches such as Bug Rocket, MATF, universal test languages, AQMS, and WSQF enhance testing efficiency, multi-platform automation, defect detection, and comprehensive quality evaluation, ultimately contributing to more reliable and maintainable software systems.

Tsuda et al. (2019) developed the Waseca Software Quality Framework (WSQF), a comprehensive framework for evaluating software quality based on Square. This framework implements numerous methods for measuring product quality and quality-in-use, as initially outlined in the Square documentation. The current state of software product quality was exposed by applying the WSQF to twenty-one commercially available, ready-to-use software products. Quality measurement value trends, inter-quality characteristic relationships, the quality-in-use relationship to product quality, and the product context relationship within application boundaries are all part of a comprehensive benchmark that emerges from this process [17].

Viriansky and Shaposhnikov (2019) revealed that the quality of the AQMS and the quality management object both contribute to the system's efficacy throughout its design phase. Both AQMS and the quality management object are seen as interconnected nd collaboratively processed sets of various types of information (software, procedures, algorithms, etc.) throughout the control object's life cycle. Particularly influential in determining the AQMS's quality are the early design phases, which define the control object's intended use and the most fundamental quality standards [18].

Jharko (2018) outside the technological process of software development; ways to get the quality of software that is needed by verifying and validating software at all stages of

its life-cycle; ways to define quality, procedures for justifying software quality, and models for analyzing system requirements in the software part [19].

Ibarra and Muñoz (2018) the importance and complexity of quality assurance efforts have expanded in tandem with the expansion and diversification of software. Although the average success rate for software projects has been 37% in recent years, according to The Chaos Report—which tracks these metrics—there has been no improvement in this area. Considering the importance of software development quality assurance activities, this study proposes a tool that promotes and facilitates the application of quality assurance methods [20].

Liu et al. (2017) proposed an avionics system test language that is applicable worldwide. The introduction of device type data and device collaboration procedures is the defining characteristic of test language. These features enable the automatic collaboration of logical test devices and jump machines, while also making test language more ubiquitous. Testing of actual avionics systems has begun at the China Academy of Electronics and Information Technology using this test language. Before automation, avionics system testing was labour-intensive and error-prone. Test language streamlines the process and makes testing more efficient [21].

Ma et al. (2016) introduced Bug Rocket, a platform for automated testing. Bug Rocket integrates mobile-ready automated testing methods with a distributed testing environment. Created a suite of automated testing tools and established a testing platform with forty of the most popular Android devices in this paper. Functional and compatibility testing are two areas where a case study demonstrates Bug Rocket's usefulness. In addition, Bug Rocket may log system logs and annotated GUI models in the event of a failure run, which helps with finding and fixing bugs [22].

Zun, Qi, and Chen (2016) introduced an innovative system for automatically testing mobile apps across multiple platforms. The Multi-platform Automatic Testing Framework (MATF) employs a test methodology based on keywords. Appium is a standard for mobile test automation, and this framework encapsulates and extends upon it. The process of managing test cases, plans, scripts, execution, and reports is all encompassed in it. The framework can automatically scan test cases and provide test scripts applicable to applications on both iOS and Android platforms [23].

Table 2 summarizes recent studies on automated testing and software quality assurance, highlighting advances in testing efficiency, multi-platform automation, and quality evaluation. Key challenges include scalability and integration, with future directions focusing on broader platform support and adaptable QA frameworks

Table 2. Comparative Analysis of Recent Automated Testing Frameworks and Software Quality Assurance Studies

Reference	Study On	Approach	Key Findings	Challenges/ Limitations	Future Directions
Tsuda et al. (2019)	Waseda Software Quality Framework (WSQF)	SQuaRE-based framework for comprehensive product and quality-in-use evaluation	Provides benchmark for software quality; reveals trends, relationships among characteristics, and product context impact	Limited to 21 commercial software products; applicability to broader software unknown	Expand framework to more software types and include dynamic quality metrics
Viriansky and Shaposhnikov (2019)	Automated Quality Management System (AQMS)	AQMS quality determined by design process; considers AQMS and management objects as	The effectiveness of AQMS depends on early design stages; establishes goals and quality requirements	Relies heavily on early design; may not adapt easily to late changes	Refine AQMS adaptability and dynamic quality assessment mechanisms

		interrelated information			
Jharko (2018)	Software quality verification and validation	Analysis of technological process violations, quality definition, and life-cycle software verification	Highlights methods for achieving required software quality through verification and validation	Conceptual; lacks specific implementation details	Develop practical tools to enforce quality at all life-cycle stages
Ibarra and Muñoz (2018)	Quality assurance tools for software development	A tool supporting implementation of QA practices	Promotes and facilitates QA practices; addresses low project success rate (avg. 37%)	Tool adoption and integration challenges in diverse software environments	Broaden tool adoption, integrate AI-based support for QA practices
Liu et al. (2017)	Standardized language for avionics system testing	Allows logical test devices and jump machines to automatically collaborate by introducing device type data and device collaboration actions.	Avionics system testing workflows are altered, and test efficiency is enhanced.	Limited to avionics system context; may need adaptation for other domains	Generalize language for broader industrial system testing
Ma et al. (2016)	BugRocket automated testing platform	Automated testing methods for mobile devices integrated into a distributed testing system	Works for functional and compatibility testing; records failed runs with annotated GUI model and system logs to aid bug fixing	Limited to the 40 most popular Android devices in the study	Extend to more devices, platforms, and broader automated testing scenarios
Zun, Qi and Chen (2016)	MATF (Multi-platform Automatic Testing Framework)	Keyword-driven test technology; encapsulates and expands Appium; integrates test case management, script generation, execution, and reporting	Can automatically parse test cases and generate scripts applicable to both iOS and Android applications	May require further optimization for scalability and complex test scenarios	Enhance multi-platform support, improve efficiency, and support more complex workflows

6. Conclusion and Future Work

Automated testing frameworks and software quality assurance practices are essential pillars in achieving robust and dependable software systems. They enable systematic defect detection, cost reduction, and improved collaboration between stakeholders across development lifecycles.

This study concludes that automated testing frameworks and tools are indispensable in modern software development, addressing critical challenges of speed, reliability, and quality assurance. While frameworks such as Keyword-Driven, Data-Driven, BDD, and Playback/Record offer unique advantages, their limitations necessitate careful selection based on project requirements. Tools like Selenium, JUnit, and TestNG further support efficient and systematic implementation of testing practices. Despite the evident benefits, challenges such as high maintenance, scalability, and incomplete requirements continue to hinder optimal outcomes. By adopting recommended best practices, including flexible

frameworks, effective test maintenance, and early defect detection, software organizations can significantly enhance product quality and stakeholder confidence.

Future research should focus on integrating AI and ML into automated testing to overcome challenges of adaptability, scalability, and predictive accuracy. AI-driven techniques such as automated test case generation, defect prediction, and real-time adaptation can reduce human intervention while improving efficiency. Expanding automation tools to support cloud-native applications, microservices, and IoT systems is another essential direction. Ultimately, empirical studies on cost-effectiveness, usability, and maintainability will inform the development of more intelligent, sustainable, and adaptable quality assurance solutions.

References

[1] M. A. Umar and C. Zhanfang, "A Study of Automated Software Testing: Automation Tools and Frameworks," *Int. J. Comput. Sci. Eng.*, vol. 8, no. 06, pp. 217–225, 2019, doi: 10.5281/zenodo.3924795.

[2] S. Chhabra and S. K. Singh, "Comparative Study and Evaluation of Web-Based Automation Testing Tools," *Int. J. Comput. Appl.*, vol. 7, no. 2, pp. 143–147, 2017.

[3] E. Alégroth, R. Feldt, and P. Kolström, "Maintenance of automated test suites in industry: An empirical study on Visual GUI Testing," *Inf. Softw. Technol.*, vol. 73, pp. 66–80, 2016, doi: 10.1016/j.infsof.2016.01.012.

[4] M. Hanna, A. Elsayed, and M.-S. M., "Automated Software Testing Frameworks: A Review," *Int. J. Comput. Appl.*, vol. 179, no. 46, pp. 22–28, 2018, doi: 10.5120/ijca2018917171.

[5] M. Hanna, A. E. Aboutabl, and M.-S. M. Mostafa, "Automated Software Testing Framework for Web Applications," *Int. J. Appl. Eng. Res.*, vol. 13, no. 11, pp. 9758–9767, 2018.

[6] S. kumar Sharma and S. Gaur, "Test Automation using Keyword Driven Approach," *Int. J. Eng. Trends Technol.*, vol. 35, no. 14, pp. 670–674, 2016.

[7] C. Chandraprabha, A. Kumar, and S. Saxena, "Data Driven Testing Framework using Selenium WebDriver," *Int. J. Comput. Appl.*, vol. 118, no. 18, pp. 18–23, 2015, doi: 10.5120/20845-3497.

[8] S. Bonfanti, A. Gargantini, and A. Mashkoor, "Generation of behavior-driven development C++ tests from abstract state machine scenarios," *Commun. Comput. Inf. Sci.*, vol. 929, pp. 146–152, 2018, doi: 10.1007/978-3-030-02852-7_13.

[9] J. Scarpino and P. Kovacs, "An Analysis Of A Software Quality Assurance Tool's Implementation: A Case Study," *Issues Inf. Syst.*, vol. 9, no. 2, 2008.

[10] S. Xu and T. Li, "Evaluation of Test-Driven Development : An Academic Case Study," *Springer*, pp. 229–230, 2009.

[11] M. Beller, R. Bholanath, S. McIntosh, and A. Zaidman, "Analyzing the state of static analysis: A large-scale evaluation in open source software," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2016*, 2016, pp. 470–481. doi: 10.1109/SANER.2016.105.

[12] M. A. Umar, "Comprehensive study of software testing: Categories, levels, techniques, and types," vol. 5, pp. 32–40, 2019, doi: 10.36227/techrxiv.12578714.

[13] S. Gojare, R. Joshi, and D. Gaigaware, "Analysis and design of selenium webdriver automation testing framework," *Procedia Comput. Sci.*, vol. 50, pp. 341–346, 2015, doi: 10.1016/j.procs.2015.04.038.

[14] M. S. Hossain, "Challenges of Software Quality Assurance and Testing," *Int. J. Softw. Eng. Comput. Syst.*, vol. 4, no. 1, pp. 133–144, 2018, doi: 10.15282/ijsecs . 4.1.2018.11.0044.

[15] M. Mothey, "Software Testing Best Practices in Large-Scale Projects," *Int. J. Sci. Res. Comput. Sci. Eng. Inf. Technol.*, pp. 712–721, 2018, doi: 10.32628/CSEIT182546.

[16] S. Perla, "Modern Practices In Software Testing And Quality Assurance," *Nanotechnol. Perceptions*, vol. 14, pp. 155–163, 2018.

[17] N. Tsuda *et al.*, "WSQF: Comprehensive Software Quality Evaluation Framework and Benchmark Based on SQuaRE," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2019, pp. 312–321. doi: 10.1109/ICSE-SEIP.2019.00045.

[18] Z. Y. Viriansky and S. O. Shaposhnikov, "Quality Assurance of Quality Management Systems," in *2019 International Conference "Quality Management, Transport and Information Security, Information Technologies" (IT&QM&IS)*, 2019, pp. 323–325. doi: 10.1109/ITQMIS.2019.8928299.

[19] E. Jharko, "Verification and Software Quality Assurance for Nuclear Power Engineering," in *2018 Eleventh International Conference "Management of large-scale system development" (MLSD*, 2018, pp. 1–4. doi: 10.1109/MLSD.2018.8551840.

[20] S. Ibarra and M. Muñoz, "Support tool for software quality assurance in software development," in *2018 7th International Conference On Software Process Improvement (CIMPS)*, 2018, pp. 13–19. doi: 10.1109/CIMPS.2018.8625617.

[21] Y. Liu, J. Lv, W. Wang, T. Li, and S. Ma, "A test language for avionics system," in *2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, 2017, pp. 28–32. doi: 10.1109/ICSESS.2017.8342857.

[22] X. Ma, N. Wang, P. Xie, J. Zhou, X. Zhang, and C. Fang, "An Automated Testing Platform for Mobile Applications," in *2016 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, 2016, pp. 159–162. doi: 10.1109/QRS-C.2016.25.

[23] D. Zun, T. Qi, and L. Chen, "Research on automated testing framework for multi-platform mobile applications," in *2016 4th International Conference on Cloud Computing and Intelligence Systems (CCIS)*, 2016, pp. 82–87. doi: 10.1109/CCIS.2016.7790229.