Review

An Overview on Testing using Selanium

Sibi Mathew *

Posted Date: 15 April 2024

doi: 10.20944/preprints202404.0911.v1

Keywords: selenium testing tool; automation-based testing; test cases; selenium IDE

 doi:10.20944/preprints202404.0911.v1

Article

An Overview on Testing Using Selanium

Sibi Mathew

MTech Scholar, Department of CSE, TKM College of Engineering, Kollam, Kerala 691005, India; 22082@tkmce.ac.in

Abstract: Among the different steps in software development software testing is one of the inevitable steps. Manual testing is considered to be more expensive and time-consuming, and this led to the formation of automated testing tools. Automated software testing tools offer a comprehensive system testing solution that minimizes the risk of any errors that could result in financial losses. One such tool is selenium. Selenium testing is an advantageous method employed by developers and testers to assess their code. Selenium is compatible with numerous widely-used programming languages like C#, Java, Python, and more. This paper focuses on understanding the in and outs of software testing using selenium

Keywords: selenium testing tool; automation-based testing; test cases; selenium IDE

1. Introduction

Software testing is an integral software engineering practice that aims to verify if the actual outcomes of the developed software correspond to the expected results. Its purpose is to ensure the software system is free from defects. Testing involves executing software or system components to assess various desired properties. Software testing can be defined as the systematic process of examining software to determine if it meets the specified requirements and to identify and detect any errors or defects that may occur within the software.

Testing can be either manual or automated. Manual testing is carried out by testers who perform the testing tasks manually, whereas automated testing relies on the assistance of specialized tools. In both cases, the identification and resolution of bugs are essential to ensure the proper functioning of the software. Testing can be conducted either through manual means or by utilizing automated techniques.

Selenium is a widely used and freely available testing tool designed to test online and real-time applications. It enables the automation of web browser interactions, allowing programming scripts to replicate manual interactions. Selenium is particularly well-suited for testers who possess coding skills and understand how to integrate different frameworks effectively. Nowadays, it is considered the de facto framework to develop end-to-end tests for web applications and supports a multi-million dollar industry[1,2].

2. concise Overview of Selenium's History

Selenium was initially introduced as an open-source project by Jason Huggins and Paul Hammant in 2004 during their tenure at ThoughtWorks. They opted for the name "Selenium" as a deliberate contrast to Mercury, an existing testing framework developed by Hewlett-Packard. Huggins envisioned Selenium as a testing framework that would overcome the limitations of Mercury. The name choice drew inspiration from the fact that selenium, a mineral, has the ability to counteract the toxicity of mercury in the body.

The original version of Selenium, referred to as Selenium Core, enabled interaction with web applications by simulating user actions such as opening URLs, clicking on links, and entering text into forms, all in an automated manner. Selenium Core functioned as a JavaScript library that interpreted a set of commands known as Selenese commands.

Selenium Core has evolved into a JavaScript library that combines the features of both Selenium Remote Control (RC) and Selenium IDE. While Selenium was widely adopted and had its advantages, it had certain limitations. Selenium's reliability was not always consistent, and executing certain tasks

 doi:10.20944/preprints202404.0911.v1

within Selenium could be challenging. As web applications became more complex and advanced, using Selenium to maintain them became increasingly difficult.[3–5]

In response to these challenges, Google developer Simon Stewart started working on improving Selenium and introduced a new approach called WebDriver. The WebDriver aimed to address the limitations of Selenium by adopting a native browser-based approach and providing a direct means of communication with web browsers. Initially, there were some misuses of Selenium within Google, but critics pointed out the limitations of the tool. Simon Stewart sought to develop a software testing system that would utilize a local software strategy and a functional framework to directly interact with web browsers. In 2008, WebDriver emerged as a significant solution to overcome the limitations of Selenium.

3. General Architecture

The architecture of the Selenium Web Tool is depicted in Figure 1 and comprises two key components: the Client and the Selenium Server. The Client includes the WebDriver API, which facilitates web page interactions and provides access to other application features. It also encompasses the Remote WebDriver class, responsible for establishing communication with the remote Selenium server.[7,8]

The Selenium Server consists of a server component that receives requests from the Remote WebDriver class in the Selenium client. Additionally, it incorporates the Application Driver API, which enables web browser testing on a server machine.

The fourth component is Selenium Grid, which is utilized by the Selenium Server through command line parameters to leverage grid capabilities. Selenium Grid consists of a central hub and multiple nodes, each possessing the capabilities of preferred browsers. This enables parallel execution of tests across multiple machines and various browsers, ultimately leading to reduced runtime.

The different components of Selenium can be described as follows:

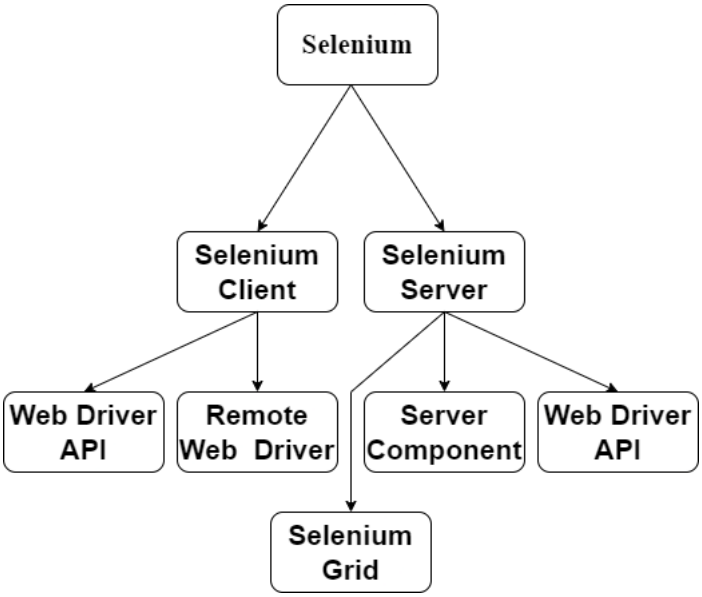

Figure 1. General Architecture

Selenium IDE is a Firefox plug-in that provides a user-friendly environment for testers to record their actions while performing tests. It allows testers to record their workflow as they interact with the application under test. The recorded actions can be replayed to automate the testing process.Selenium IDE also includes an integrated development environment (IDE) where testers can develop and enhance their test cases. The IDE offers a range of features such as editing, debugging, and organizing test cases. Testers can utilize Selenium IDE to build robust and comprehensive test cases that cover various scenarios and interactions within the application.

 doi:10.20944/preprints202404.0911.v1

Selenium Remote Server, also known as Selenium Grid, is a component of Selenium that enables distributed testing across multiple machines and browsers. It allows testers to execute their tests in parallel on different machines, facilitating faster test execution and improved efficiency.

The Selenium Remote Server operates as a hub and node system. The hub acts as a centralized server that manages and delegates test requests to the available nodes. The nodes, on the other hand, are individual machines or devices that have different browsers and operating systems configured for testing.The Selenium Remote Server receives requests from the Selenium client, which may include test scripts developed using Selenium WebDriver. It then distributes these test requests to the appropriate nodes based on the desired browser and platform configurations.By utilizing Selenium Remote Server, testers can take advantage of the distributed testing capability to run tests simultaneously on multiple machines and browsers. This helps in achieving better test coverage, reducing test execution time, and improving overall test efficiency.

Selenium WebDriver is a powerful and widely used tool in the Selenium suite. It provides a programming interface for interacting with web browsers and automating web application testing. WebDriver allows testers and developers to write code in various programming languages (such as Java, C#, Python, etc.) to automate browser actions, perform validations, and manipulate web elements.[9,10]

Key features and functionalities of Selenium WebDriver include:

1. Parallel test execution: Selenium Grid allows for simultaneous execution of tests on multiple machines and browsers, distributing the workload and reducing overall test execution time.
2. Cross-browser and cross-platform testing: With Selenium Grid, tests can be executed on different browsers (such as Chrome, Firefox, Safari, Internet Explorer) and operating systems (Windows, macOS, Linux). This helps ensure compatibility and validate the behavior of web applications across various configurations.
3. Hub and node architecture: Selenium Grid operates on a hub and node system. The hub acts as the central control point that receives test requests and delegates them to available nodes. The nodes are individual machines or devices with different browser configurations.
4. Scalability and resource optimization: By leveraging Selenium Grid, testers can utilize available machines or devices as nodes, enabling efficient utilization of resources and scaling test execution based on the testing needs.
5. Remote WebDriver support: Selenium Grid is compatible with Selenium WebDriver, allowing testers to use the same WebDriver API for interacting with browsers across the grid infrastructure.
6. Test distribution and load balancing: Selenium Grid automatically distributes test requests across available nodes, ensuring an even workload distribution and efficient utilization of resources. It can also perform load balancing, optimizing the test execution process.

By leveraging Selenium WebDriver, testers and developers can automate the testing of web applications, enhance test coverage, improve accuracy, and speed up the overall testing process.

Selenium Grid is indeed a tool that enables the parallel execution of test cases across multiple machines. By distributing the workload and running test cases simultaneously on different machines, Selenium Grid significantly reduces the overall execution time of test suites. This parallel execution capability helps improve testing efficiency, especially when dealing with a large number of test cases or when targeting different browser configurations. Ultimately, Selenium Grid contributes to faster test execution and quicker feedback on the application's behavior across various environments.

4. General Testing Using Selenium

- Test Planning: Identify the testing objectives, scope, and requirements. Determine the browsers and platforms to be tested and prioritize test cases.[12]
- Test Environment Setup: Set up the test environment by installing the necessary software components, including Selenium WebDriver, relevant browser drivers, and any additional dependencies.

 doi:10.20944/preprints202404.0911.v1

- Test Case Design: Design test cases based on the identified requirements and objectives. Define test scenarios, test data, and expected outcomes.
- Test Script Development: Write test scripts using the programming language supported by Selenium WebDriver (such as Java, C#, Python). Use WebDriver APIs to interact with web elements, perform actions, and validate results.
- Test Execution: Execute the test scripts on the target browsers and platforms. Use Selenium WebDriver to launch the browser, navigate to the application, perform actions, and verify the expected outcomes.
- Test Result Analysis: Analyze the test results to identify any failures or issues. Log defects in a bug tracking system and investigate the root causes of failures.
- Test Reporting: Generate test reports to document the test execution results. Include information such as test coverage, passed tests, failed tests, and any defects found.
- Test Maintenance: Update test scripts and test cases as necessary to accommodate changes in the application or requirements. Maintain test data and keep the test environment up to date.
- Test Automation Framework Integration: Integrate Selenium tests with a test automation framework, such as JUnit or TestNG, for enhanced test management, reporting, and scheduling.

5. Key Test Factors

Key test factors, also known as testing factors or testing dimensions, are the aspects that influence the testing process and contribute to the effectiveness and efficiency of testing efforts.

5.1. Test Script Development

Test Script Development in Selenium involves writing code to automate interactions with web elements, perform actions, and validate expected outcomes. Here are the key steps involved in test script development:

1. Set up the Development Environment: Install the required programming language and Selenium WebDriver. Set up the development environment with the necessary IDE or text editor.
2. Identify Test Scenarios: Analyze the application requirements and identify the test scenarios to be automated. Break down the scenarios into individual test cases for scripting.
3. Define Test Data: Identify the necessary test data required for each test case. This includes input data, expected outcomes, and any preconditions or prerequisites.
4. Choose the Programming Language: Select the programming language for scripting, such as Java, C#, Python, etc. Ensure that the chosen language is compatible with Selenium WebDriver.
5. Write Test Scripts: Use the programming language and the Selenium WebDriver APIs to write test scripts. Start by setting up the WebDriver instance, launching the browser, and navigating to the application under test.
6. Interact with Web Elements: Use WebDriver methods to locate and interact with web elements. Perform actions such as clicking buttons, filling out forms, selecting options from dropdowns, and handling checkboxes.
7. Validate Results: Use assertions and verification techniques to validate the expected outcomes. Compare actual results with the expected values or conditions and log any discrepancies.
8. Handle Synchronization: Implement appropriate synchronization techniques to handle dynamic elements, delays, or asynchronous behavior in the application. Use techniques like waits, explicit waits, or expected conditions to ensure accurate test execution.
9. Implement Error Handling: Incorporate error handling mechanisms to handle exceptions or unexpected behavior during test execution. Use try-catch blocks or exception handling techniques to capture and handle errors gracefully.
10. Organize and Maintain Test Scripts: Follow best practices to organize and maintain test scripts effectively. Use modularization, functions, or Page Object Model (POM) to enhance reusability and maintainability.

 doi:10.20944/preprints202404.0911.v1

11. Execute and Debug Test Scripts: Execute the test scripts and validate their functionality. Use debugging techniques to troubleshoot any issues encountered during execution.
12. Continuous Integration: Integrate the test scripts into a continuous integration system or test automation framework, such as JUnit or TestNG, for streamlined execution, reporting, and management.

By following these steps, testers can develop robust and reliable test scripts using Selenium WebDriver, enabling efficient automation of test scenarios and facilitating effective web application testing.

5.2. Test Execution and Parallel Testing

Test Execution and Parallel Testing in Selenium involve executing test cases simultaneously on multiple machines or browsers to speed up the overall testing process and improve efficiency. Here are the key aspects to consider:

1. Selenium Grid: Selenium Grid is a component of Selenium that enables parallel test execution across multiple machines or virtual environments. It consists of a hub and nodes, where the hub acts as the central control point and the nodes represent the individual machines or browsers available for testing.
2. Test Distribution: With Selenium Grid, test cases are distributed among the available nodes for execution. Each node can handle a specific browser or operating system configuration, allowing for simultaneous execution across different environments.
3. Scalability: Selenium Grid provides scalability by leveraging the available resources efficiently. By utilizing multiple machines or virtual environments, it enables the testing of a large number of test cases or scenarios simultaneously, reducing the overall execution time.
4. Cross-Browser Testing: Parallel testing with Selenium Grid is particularly useful for cross-browser testing. It allows running test cases across different browsers (such as Chrome, Firefox, Safari, Internet Explorer) concurrently, ensuring compatibility and consistent behavior across multiple platforms.
5. Reduced Execution Time: By executing test cases in parallel, Selenium Grid significantly reduces the overall execution time. This leads to faster feedback on the application's behavior and enables quicker identification of issues or defects.
6. Test Result Consolidation: Selenium Grid provides mechanisms to consolidate test results from multiple nodes. This allows testers to view the combined test results and analyze the overall outcome of the parallel test execution.
7. Test Stability and Isolation: When executing tests in parallel, it is essential to ensure test stability and isolation. Test cases should be designed in a way that they do not interfere with each other, and dependencies or conflicts between tests should be managed effectively.
8. Reporting and Analysis: Selenium Grid can generate consolidated test reports that provide insights into the overall test execution status, including passed tests, failed tests, and any errors encountered during parallel testing.

   By leveraging Selenium Grid for parallel test execution, testers can significantly accelerate the testing process, increase test coverage, and improve efficiency in identifying issues across different browsers or environments. It enables effective utilization of resources and facilitates faster feedback on the application's behavior.

5.3. Integration with Test Automation Frameworks

Integration with Test Automation Frameworks is a crucial aspect of Selenium testing to enhance test management, reporting, and overall test automation capabilities. Here are the key points to consider:[14,15]

1. Test Automation Frameworks: Test automation frameworks provide a structured approach to organizing and executing automated tests. Examples of popular test automation frameworks

 doi:10.20944/preprints202404.0911.v1

include JUnit, TestNG, NUnit, and PyTest. These frameworks offer various features such as test case management, test data management, reporting, and test execution control.
2. Test Case Organization: Integration with a test automation framework helps in organizing test cases effectively. Test cases can be grouped into test suites or test classes based on functionalities, modules, or scenarios. This allows for better test case management and easy execution.
3. Test Execution Control: Test automation frameworks offer control over test execution, enabling the selection of specific test cases or test suites for execution. Test runners provided by the frameworks facilitate executing tests with different configurations, such as running specific tests in parallel, executing tests in a specific order, or running tests on different environments.
4. Test Data Management: Test automation frameworks provide mechanisms for managing test data. They enable the separation of test data from test logic, allowing for reusable and maintainable test scripts. Test data can be provided through configuration files, databases, or data-driven approaches, enabling efficient data management for different test scenarios.
5. Reporting and Logging: Integration with a test automation framework enhances reporting capabilities. Detailed test execution reports can be generated, including information on test pass/fail status, execution time, and any captured errors or exceptions. Customized reports can be generated for different stakeholders, facilitating better analysis and decision-making.
6. Assertions and Assertions Libraries: Test automation frameworks often provide built-in assertion libraries or assertion capabilities. These libraries help in verifying expected outcomes and comparing actual results with expected values. They offer a wide range of assertion methods and assertions customization options to handle different validation scenarios.
7. Continuous Integration (CI) Integration: Test automation frameworks seamlessly integrate with continuous integration systems, such as Jenkins, Bamboo, or Azure DevOps. This integration allows for scheduled or triggered test execution as part of the CI/CD pipeline, ensuring continuous testing and rapid feedback on application quality.
8. Test Configuration Management: Test automation frameworks often provide features for managing test configurations, such as environment-specific configurations, browser configurations, or test environment setup. This enables easy switching between different configurations and ensures test consistency across various environments.

   Integration with a test automation framework streamlines the management, execution, and reporting of Selenium tests. It enhances collaboration among team members, improves test efficiency, and provides robust test automation capabilities for achieving reliable and maintainable test suites.

6. Community Support and Resources

Community support and resources play a crucial role in Selenium testing, providing valuable knowledge, assistance, and learning opportunities. Here are the key aspects to consider:

1. Online Communities and Forums: Selenium has a vibrant community of users, testers, and developers who actively participate in online communities and forums. Platforms like Stack Overflow, Reddit, and Selenium official forums allow users to ask questions, seek help, and share their experiences. Engaging with these communities can provide valuable insights, solutions to problems, and access to best practices.
2. Official Documentation: Selenium offers comprehensive official documentation that includes user guides, API references, and tutorials. The documentation covers various aspects of Selenium, including setup, usage, best practices, and advanced features. It serves as a valuable resource for understanding Selenium's capabilities and effectively utilizing its features.
3. Online Tutorials and Blogs: Numerous tutorials and blog posts are available online that provide step-by-step guidance, tips, and examples for Selenium testing. These resources are often created by experienced testers and developers who share their knowledge and insights. They cover a wide range of topics, from basic test script development to advanced techniques and framework integration.

 doi:10.20944/preprints202404.0911.v1

4. Webinars and Online Courses: Webinars and online courses are offered by Selenium experts, testing organizations, and educational platforms. These provide structured learning opportunities, covering various aspects of Selenium testing in-depth. Webinars often include live demonstrations, interactive Q&A sessions, and practical examples to enhance understanding and skills.
5. Open-Source Collaboration: Selenium being an open-source project encourages collaboration and contribution from the community. Users can contribute to the development of Selenium by reporting bugs, suggesting enhancements, or submitting code contributions. Engaging in open-source collaboration provides an opportunity to interact with experienced developers and gain deeper insights into Selenium.
6. Social Media Groups: Social media platforms, such as LinkedIn and Twitter, host groups and communities dedicated to Selenium testing. Joining these groups allows users to connect with like-minded professionals, stay updated with the latest trends, and access additional resources shared by community members.
7. Selenium Conferences and Events: Selenium conferences and events are organized globally, bringing together Selenium enthusiasts, experts, and industry professionals. These events feature keynote speeches, workshops, and presentations on various Selenium-related topics. Attending conferences provides opportunities for networking, knowledge sharing, and staying updated with the latest advancements in Selenium testing.
8. Selenium WebDriver GitHub Repository: Selenium WebDriver's official GitHub repository is an essential resource for accessing the source code, documentation updates, and issue tracking. Users can explore the repository to understand the latest developments, review existing issues, and contribute to the project.

   Engaging with the Selenium community and leveraging available resources not only expands knowledge but also provides valuable support in overcoming challenges, improving test practices, and staying up-to-date with emerging trends in Selenium testing.

7. Limitations and Challenges

Selenium testing, like any other testing approach, has certain limitations and challenges. It's important to be aware of these to effectively plan and execute Selenium tests. Here are some common limitations and challenges associated with Selenium:

- Limited Support for Desktop Applications: Selenium primarily focuses on web-based applications and lacks comprehensive support for testing desktop applications. While there are some workarounds available, Selenium's core functionality is designed for web testing.
- Cross-Domain Security Restrictions: Due to browser security restrictions, Selenium faces challenges when interacting with elements or executing tests across different domains. This can limit the scope of testing for applications that rely heavily on cross-domain interactions.
- Complex Test Maintenance: As web applications evolve, test scripts developed using Selenium may require frequent updates to accommodate changes in the application's UI or functionality. Test maintenance can become time-consuming, especially for large test suites or applications with frequent updates.
- Lack of Built-in Reporting: Selenium does not provide built-in reporting capabilities. While test execution results can be logged and captured, generating comprehensive reports with detailed insights may require additional tools or custom development.
- Dependency on Browser Versions: Selenium's compatibility with different browser versions is critical for successful test execution. Browser updates or changes in browser behavior can sometimes lead to compatibility issues, requiring updates or adjustments in the Selenium scripts.
- Time and Effort in Test Script Development: Creating effective and robust test scripts using Selenium requires a solid understanding of the framework and programming languages. Test script development can be time-consuming, especially for testers who are new to Selenium or have limited programming knowledge.

 doi:10.20944/preprints202404.0911.v1

- Limited Support for Mobile Applications: Selenium WebDriver primarily focuses on web-based testing and has limited support for mobile application testing. While there are frameworks like Appium for mobile testing, integrating them with Selenium can add complexity to the testing process.
- Performance Testing Limitations: Selenium is primarily designed for functional testing and may not provide comprehensive performance testing capabilities. Dedicated performance testing tools may be required to assess the application's performance under different load conditions.
- Test Execution Speed: Selenium tests can sometimes be slower compared to manual testing, especially when running test cases sequentially. Parallel test execution using Selenium Grid can help mitigate this limitation to some extent.
- Learning Curve and Skill Requirements: Selenium requires knowledge of programming languages, frameworks, and web technologies. Testers without programming experience may face challenges in effectively utilizing Selenium's features and addressing complex testing scenarios.

  While Selenium is a powerful testing tool, it's essential to consider these limitations and challenges while planning and executing tests. Being aware of these factors can help testers make informed decisions, optimize their test strategies, and explore additional tools or frameworks to overcome specific challenges.

8. Conclusions

In conclusion, Selenium is a powerful and widely used testing tool that offers numerous benefits for software testing. It provides automation capabilities, cross-browser compatibility, and supports multiple programming languages, making it convenient for developers and testers. Selenium Core, Selenium WebDriver, and Selenium Grid are the key components that enable efficient test script development, execution, and parallel testing across different machines.

However, it's important to be aware of the limitations and challenges associated with Selenium. It may have limited support for desktop applications, cross-domain security restrictions, and complex test maintenance requirements. Additionally, Selenium's compatibility with browser versions, the need for test script development skills, and its focus on web-based testing should be taken into consideration.

Nevertheless, the Selenium community and available resources provide immense support for testers. Online communities, forums, documentation, tutorials, and social media groups offer a wealth of knowledge and assistance. Engaging with the community, attending conferences, and leveraging online courses can enhance skills and keep testers updated with the latest trends.

Incorporating Selenium into a test automation framework, considering key test factors such as functionality, usability, performance, reliability, compatibility, security, maintainability, testability, scalability, and compliance, will contribute to comprehensive and effective testing.

Overall, Selenium is a valuable tool for automating software testing, reducing testing costs, and improving test efficiency. With proper understanding, planning, and utilization, Selenium can significantly contribute to the success of software testing efforts.

References

1. R. Abbas, Z. Sultan, and S. N. Bhatti, "Comparative analysis of automated load testing tools: Apache JMeter, Microsoft Visual Studio (TFS), LoadRunner, Siege," 2017 International Conference on Communication Technologies (ComTech), pp.39-44, 2017.
2. Cerioli, M.; Leotta, M.; Ricca, F. What 5 million job advertisements tell us about testing: A preliminary empirical investigation. In Proceedings of the 35th Annual ACM Symposium on Applied Computing, Brno, Czech Republic, 30 March–3 April 2020; pp. 1586–1594.
3. McMahon, C, "History of a Large Test Automation Project Using Selenium", 2009.8.
4. Antawan Holmes and Marc Kellogg, Automating Functional tests using Selenium, Proceedings of AGILE 2006 Conference, 2006.

 doi:10.20944/preprints202404.0911.v1

5. Vila, E.; Novakova, G.; Todorova, D. Automation testing framework for web applications with Selenium WebDriver: Opportunities and threats. In Proceedings of the International Conference on Advances in Image Processing, Bangkok, Thailand, 25–27 August 2017; pp. 144–150.
6. García, Boni, et al. "A survey of the selenium ecosystem." Electronics 9.7 (2020): 1067.
7. Nyamathulla, S., P. Ratnababu, and Nazma Sultana Shaik. "A Review on Selenium Web Driver with Python." Annals of the Romanian Society for Cell Biology (2021): 16760-16768.
8. Alferidah, Saja Khalid, and Shakeel Ahmed. "Automated software testing tools." 2020 International Conference on Computing and Information Technology (ICCIT-1441). IEEE, 2020.
9. Haugset, Børge, and Geir Kjetil Hanssen. "Automated acceptance testing: A literature review and an industrial case study." Agile 2008 Conference. IEEE, 2008.
10. Alenzi, Abdullah, et al. "A survey of software testing tools in the web development domain." Journal of Computing Sciences in Colleges 38.2 (2022): 63-73.
11. Kuutila, M. Benchmarking Configurations for Web-Testing-Selenium Versus Watir. Master's Thesis, Faculty of Information Technology and Electrical Engineering, University of Oulu, Oulu, Finland, 2016.
12. Leotta, M.; Clerissi, D.; Ricca, F.; Spadaro, C. Comparing the maintainability of Selenium WebDriver test suites employing different locators: A case study. In Proceedings of the 2013 International Workshop on Joining Academia and Industry Contributions to Testing Automation, Lugano, Switzerland, 15–20 July 2013; pp. 53–58.
13. Kuutila, M. Benchmarking Configurations for Web-Testing-Selenium Versus Watir. Master's Thesis, Faculty of Information Technology and Electrical Engineering, University of Oulu, Oulu, Finland, 2016.
14. Presler-Marshall, K.; Horton, E.; Heckman, S.; Stolee, K. Wait, Wait. No, Tell Me. Analyzing Selenium Configuration Effects on Test Flakiness. In Proceedings of the 2019 IEEE/ACM 14th International Workshop on Automation of Software Test (AST), Montreal, QC, Canada, 27 May 2019; pp. 7–13.
15. Deepthi Wilson. R, Manjuprasad. B, "A Compressive Review on selenium Automation Testing Tools", Department of Computer Science & Engineering, GSSSIETW, Mysuru, IJERT, ISSN: 2278-0181 2017.
16. Satish Gojarea,*, Rahul Joshib,Dhanashree Gaigawarec, "Analysis and design of selenium web driver automation testing framework", 2nd International Symposium on Big Data and Cloud Computing (ISBCC'15): 1877-0509 © 2015
17. JyotiDevi, Kirti Bhatia, Rohini shrama, "A Study on Functioning of Selenium Automation Testing Structure", International Journal of computer science and software engineering.and Knowledge Management, may 2017, Volume 7, issue-5 No. 2, pp. 855-862, ISSN: 2277 128X
18. Srashti Lariya1 Dr. Sameer Shrivastava2 Er. Sumit Nema3, "Automation Testing using Selenium Web Driver & Behavior Driven Development (BDD)", IJSRD - International Journal for Scientific Research & Development| Vol. 6, Issue 03, 2018 | ISSN (online): 2321-0613
19. Lokaiah Pullagura, Dr. Anil Kumar, "An Effective LSTM Network Model For Accurate Prediction of Delays in Indian Railway Networks", International Journal of Advanced Trends in Computer Science and Engineering, Vol 9, No.4, July-August 2020.