SSRG International Journal of Electrical and Electronics Engineering
ISSN: 2348-8379/ https://doi.org/10.14445/23488379/IJEEE-V12I2P113

Volume 12 Issue 2, 113-123, February 2025

Review Article

Comparative Review on Automated Test Failure Detection and Healing Tools

Nammi Hemanth Kumar[1 *], Sireesha Rodda[2]

[1,2]*Department of Computer Science Engineering, GITAM (Deemed to be University), Visakhapatnam, Andhra Pradesh, India.*

[*]*Corresponding Author : hnammi@gitam.in*

Received: 11 December 2024 Revised: 10 January 2025 Accepted: 08 February 2025 Published: 22 February 2025

Abstract *- The main aim of this paper was to evaluate automated test failure detection and healing tools in software test automation. Although Artificial Intelligence and Machine Learning involve creating separate and individual algorithms for accessing data and making sense of it by identifying patterns to form conclusions, these predictions should be used to their full benefit for software testing. Automated test failure detection and healing tools are one approach that makes more of these predictions become a reality under software testing. This paper reviews the existing literature regarding healing tools specifically created for test failure detection and healing, particularly their performance in recognizing User Interface changes and healing the test scripts automatically. The review presents the key characteristics, features, functionalities, and technologies used in the tools, such as Artificial Intelligence, machine learning, visual testing, and integration with popular test automation frameworks. By juxtaposing the sources reviewed above, the review outlines the pros and cons and promising application areas of each and provides suggestions for appropriate uses in highly diverse testing conditions and contexts. Moreover, the review also starts with the gaps and the challenges that the current cutting-edge approaches have faced and gives a future outlook on what directions future research and development have in terms of automated test failure detection and healing. Somewhere It seems like there is no distinctive technique framework or tool available that could support the automated test failure detection and healing and can fulfill all the requirements. Finally, this paper ends with a discussion of the most popular tools available, along with the expressed thought process about the present and forthcoming artificial intelligence for test automation.*

Keywords *- Test automation**,** Artificial Intelligence, Machine Learning, Self-healing tools.*

1. Introduction

Testing is a necessary part of the process of constructing any software. It ensures that the software produced is reliable and responsive to users' needs [1]. Testing can prevent expensive problems down the line by catching errors expeditious in the development process. Different tests can be used, and which ones will depend on the particular software that is being developed. Nevertheless, the ultimate aim is to locate defects in the software so that they can be repaired before it is released to users. These defects can be slight bugs or faults, meaning errors in the code itself. However, some can be serious problems that might cause the software to fail or crash.

Last but not least, finding and repairing defects early can lower the cost of correcting them later [2]. Manual and automation testing are other primary divisions in software testing. The former refers to software application testing whereby it's testing stands in the hands of a human being's actions, while the latter is a software tool or program that controls the execution of tests, comparing the pages and results produced by a program under test with the expected behavior and mark it as passed or fail [3]. A manual tester enters data into the application and uses the application by interfacing with the application and checking how the application responds. Automation testing refers to testing software applications by employing a software tool that executes the application's source code. The test, most likely, is written by an Automation tester who, in advance, determines the actions that the tool will perform to test the software's source code [4]. As test automation in software development continues to grow, robust and reliable test execution has become more important.

The traditional automation tools are based on static scripts, which are extremely sensitive to any changes in the AUT. A slight modification of the UI can cause the failure of tests if element attributes have been changed, the layout is modified, or dynamic identifiers are introduced, necessitating frequent script maintenance. Self-healing automation tools aim to address this issue by automatically detecting and adapting to changes in the AUT. While promising, current solutions lack robustness, often failing in highly dynamic environments or introducing inefficiencies.

1.1. Software Testing

Software testing is known as the way to find defects, while software testing has different reasons for conducting it. One reason is improved software quality, where user requirements and expectations are checked for this product so that it can be said to be high-quality software. Testing ensures the smooth running of a software system. In Software Developing Life Cycle (SDLC), software-developing companies spend much effort and time on testing [5]. Early detection of defects at the SDLC saves time and money, but if they are discovered later on during development, there will be a significant rise in both time-to-market and costs.

Consequently, performing the test across all stages of SDLC is more fruitful in identifying the flaws in the program [6]. It is more cost-effective to fix them earlier before release. Software testing aims at the evaluation of applications capabilities or products, for example, reliability, portability, efficiency, security, and usability, among others, which should be thoroughly tested by checking out all these principles [7]. The intent of software testing is to detect errors or defects and avoid the recurrence of defects in the software, which aftermath in the overall improved effectiveness of the system.

1.2. Manual Software Testing

This is the most basic level of testing, where test cases are executed directly by interacting with the software. The tester creates a set of test examples that illustrate the features and desired output of the software to be tested. These test cases are language natural plain text. It is time-consuming because, in manual testing, every activity will be performed manually by the tester. It is more cost-effective, but it may be a good choice in the case of some complex systems where critical issues are not likely to be found via automated testing. In manual testing, the tester plays a key role as the end user and checks all features of the software to make sure that the behavior is intact [8].

1.3. Automated Software Testing

The process of testing becomes efficient. Automated software testing makes it easy to run different tests, such as regression tests and performance tests. The advent of automated testing made hard testing activities easier because it can test a wide range of data sets and also duplicate the tests many times without any human intervention. Automated software testing calls for minimal capital outlay in terms of purchasing licensed tools; however, this is insignificant as compared to the cost savings realized from reduced efforts in manual software testing [4].

Some phases through which automated software testing passes are developing the test plan or preparing the test cases, opting for the best tool to use, developing the scripts, and finally running the automated testing tool using the developed script. Automating software tests aims to reduce both the time taken and the cost involved in such an activity. This leads to more efficient operation in relation to reducing human involvement during a test process. Testing automation supports reusability for different upgrades of the tested system by employing tester's tool scripts [9]. There are several benefits to using automation testing, including:

1.3.1. Reduced Time to Test

Automation testing can reduce the time taken to test any software application. This is because automation tests can be rerun and run concurrently, thus reducing the time spent on testing.

1.3.2. Improved Accuracy

Automation tests improve the precision of testing. This occurs since automation tests are performed similarly every time, resulting in consistent outcomes.

1.3.3. Reduced Costs

Automation testing reduces costs during testing. It means that automation tests can reduce labor costs.

If you are thinking about automated testing, then you have an array of tools from which to choose [10]. In opting for a tool for automation testing [11], some considerations should be put in mind, such as the type of applications that need to be tested, the programming languages used, and the allocated budget. Self-healing test automation is a revolutionary improvement in software testing [12], which aims at addressing the problem of test automation scripts becoming outdated when application interfaces change by minimizing the necessity of frequent manual script updates through the use of advanced technologies that enhance the endurance and productivity of automated testing. In this exploration, key attributes, uses, and benefits of these tools will be discussed. Robust testing methodologies are essential in a dynamic field like software development to ensure the quality and reliability of software applications. One crucial step has been taken through the emergence of automation testing, where companies can quickly detect bugs and validate functionality.

However, it becomes more difficult as projects get more complex over time to manage many complex and extensive test suites. The most challenging aspect is that testing teams must quickly catch and resolve test failures. Some examples of these include changes to an application's code, updates in environment configurations, or even abrupt shifts in user interface elements [13]. Untreated test failures can delay development processes and software releases and break trust in its quality.

As a result, Automated Test Failure Detection and Healing Tools have been rising [14]. These benefits are from the latest techniques like intricate algorithms, artificial intelligence, machine learning, etc. It can immediately highlight what is failing during discovery testing and requires to be fixed [15] using automatic test failure detection and

recovery tools to make the software product quality better because it helps in avoiding production bugs in case the defect reaches pro when it comes to auto-detect failures of tests automatically using some tool manage to recover them too.

Furthermore, it reduces costs associated with testing and also shortens its period while increasing testers' productivity since it eliminates the time used when downtime is inevitable for isolating issues with components of a system, thereby enhancing the efficiency of the tests. They typically employ both static analysis and dynamic analysis. Some static analysis tools scan the code for errors, while some dynamic analysis tools execute tests and check the codes against unexpected results. It has its own pros and cons as there are multiple tools that have been published. It allows you to diagnose and resolve test failures more quickly and efficiently, reducing the time spent maintaining test suites.

2. Review of Related Literature

The integration of Artificial Intelligence (AI) and Machine Learning (ML) technologies into the software testing process has been in the spotlight over the past few years. Software testing has traditionally been linked with being time-consuming and manpower-intensive, with a lot of human effort going into test case development, test execution, and result analysis. The advent of AI and ML, however, is bringing about a paradigm shift in the software testing process, leading to increased efficiency, accuracy, and overall coverage.

A study titled "Self-Healing Test Automation Framework using AI and ML" presents case studies demonstrating the application of self-healing mechanisms in various software environments. The research focuses on developing automated recovery processes that dynamically adjust tests based on real-time data, addressing technical challenges such as AI integration and scalability [16]. Here are several ways AI/ML are impacting software testing:

2.1. Test Case Generation and Optimization

Automated test cases generation and optimization by AI/ML The use of AI to sift through historical test data and code changes, AI algorithms can suggest the most critical scenarios in a shorter duration of time, making sure there are no overlaps too, which provides non-redundancy along with complete coverage [17]. It uses machine learning algorithms to understand the codebase, user stories, and requirements before automatically creating test cases. These algorithms find different paths across the code and write tests to ensure all these are covered efficiently.

Leveraging this information with machine learning models can optimize existing test suites for dormant code paths by pruning redundant or less critical test cases and thereby reducing the total number of tests. This optimization will decrease the execution time and amount of resources needed for testing. For example, a machine learning algorithm can determine and provide test cases for corner cases that were overlooked during the manual creation of the edge cases. This methodical approach also leads to hunting phased bugs and ensuring the high quality of software.

2.2. Test Execution

Maintenance of test scripts is one of the biggest challenges of automation testing. These tools leverage artificial intelligence to detect such kinds of application changes made in the Application Under Test (AUT), like UI modifications or behind-the-scene code updates and make corresponding repairments. This self-healing feature helps radically reduce the maintenance effort required to keep test scripts working over a longer period. Based on the last code changes, historical test data, and even weighing if you need to maximize QAs, their capacity or chances of detection defects in real production, AI algorithms can decide which sequence will be the most impact target for running executions.

This little trick results in running the most important tests first, thus saving time and resources. AI-Augmented Test Automation Test automation tools can execute tests much faster while beating the quality of manual testing. These tools are able to auto-adjust with any changes in the application interface or code, making it easy for developers and reducing the overhead of maintaining test scripts [18].

2.3. Defect Prediction and Analysis

By predicting defects, the testing efforts can be arranged in order of predicted risk. Testing is focused on high-risk areas identified by AI models to ensure critical bugs are captured early during development. It will detect any anomalies in the changes made to code, artifacts created during build or test results that might indicate possible defects. Such anomalies, which could go unnoticed when investigated manually, may indicate concealed issues that should be examined further. ML algorithms can analyze defect data to predict potential bugs in new code, helping developers address issues before they become critical. This proactive approach enhances software quality and reliability [19].

2.4. Regression Testing

This is done by machine learning models determining the impacts of code changes on different parts of an application. AI understands how the different pieces relate with one another and, therefore, can tell which sections may be affected as a result of recent changes requiring proper regression tests to be conducted. Machine-learning-based tools automatically update and repair test scripts that break due to changes in an application's functionality. This self-repairing capacity reduces maintenance overheads and ensures continued functionality of scripts over time. AI/machine learning knows which part of software needs retesting after a change, thus optimizing regression testing efforts. This reduces the time

spent on regression testing and ensures that new changes do not introduce new bugs [20].

2.5. Natural Language Processing (NLP) for Requirements Analysis

NLP algorithms can be used to extract key information automatically from several sources of requirements documentation, including user stories, functional specifications, and business requirement documents. Because of this extraction process, major aspects such as features, limitations and the user's desires are identified without any manual intervention. In addition, it can note inconsistencies in different requirements or gaps that lack necessary information. Thereby ensuring that a full set of requirements will be covered for the majority of key questions related to the system. [21].

2.6. Self-Healing Test Automation

Traditional test scripts are designed using static locators such as element IDs or XPaths for UI element identification and interaction purposes. When the UI changes, those locators may change, resulting in a failure. Broader test coverage is allowed by self-healing since tests are going to work perfectly even with applications that are constantly changing. It provides accurate test results and helps locate actual defects more quickly, rather than false positives due to minor UI changes. More importantly, self-healing test automation enables the maintenance of tests with script maintenance-testers can focus more on developing new scripts and less on fixing the current ones, thereby enhancing test coverage.

The self-healing system is able to adapt manually to any changes that must be applied to guarantee the reliability of the test suites, even if it transforms the application further. This process allows learning over time without always needing to go a step further and intervene manually each time, thus making this continuing education process a good tool in system monitoring. AI/ML-powered self-healing capabilities ensure the automatic detection of changes in the application under test and the adjustment of corresponding test scripts. Thus, it reduces maintenance pressure on testers, who have to cope with frequent modifications happening within applications [22].

2.7. Visual Testing

Computer vision technologies equip AI tools for visual testing. It ensures that the UI will look/behave as expected across different devices and screen resolutions. This technology is instrumental in ensuring that layouts, colors, fonts, and images appear correctly on various devices' screens, regardless of which browser a user chooses and screen resolution. Visual testing complements functional testing by guaranteeing that the application works correctly and looks like it should. This is especially important in rich graphical interface applications [23].

2.8. Intelligent Test Data Generation

Test case generation is a crucial phase of software testing. It is done to ensure that the software can be tested with different inputs and situations, thus being able to handle any unexpected inputs gracefully. Nonetheless, manual generation of test cases may be time-consuming and prone to errors. Intelligent Test Data Generation (ITDG), on the other hand, involves using artificial intelligence to automatically generate test cases. The scale of the data to be generated may differ substantially. This can make ITDG tools struggle to produce valid yet challenging data. This must involve generating very wide-ranging data that will be used to thoroughly prove the program. Thus, ITDG tools may have challenges in providing full coverage for all combinations of inputs and conditions since this would require coming up with many varied datasets. As a result, test data generation becomes faster, easier and more reliable. AI could create reliable test data that mirrors real environments and diverse edge cases, thus enabling robust tests. This is key to ensuring the integrity and security of data [24].

2.9. Root Cause Analysis

Automating the analysis process will easily spot and fix defects; this is a very big advantage because it saves the time that should have been spent in debugging and resolution process of defects, and ML algorithms can analyze large datasets with high accuracy, which minimizes human errors, therefore, increasing the trustworthiness of root cause identification. It can also trace the flow of data and control through the application to point out exactly where the code failed. AI understands code dependencies and relationships, thereby predicting the effect of a defect by analyzing affected areas and locating the root cause. Root cause patterns like defects and test failures and their graphical representation have become clearer using AI/ML. This can rapidly diagnose and resolve troubles, thereby enhancing overall development efficiency [25].

Currently, automating test generation and execution is a highly desirable improvement in testing productivity and cost reduction [26]. Automation facilitates more test cycles due to repetitive tests and more continual test runs. There are some frameworks, such as TestNG and BDD [27]. These frameworks enable the tester to write their test cases more productively and effectively in a shorter amount of time. Adding artificial intelligence to automation that is currently based on Selenium would make the testing processes more practical, adaptable, and intelligent.

Automation testing is an indispensable part of modern software testing, enabling faster releases, higher quality products, and more efficient use of resources [28]. By leveraging automation, organizations can enhance their testing processes, achieve better coverage, and ensure that their software meets the highest standards of quality and performance. This technique would enable Selenium to adapt

to the rapidly changing characteristics of today's 21st-century web applications. More importantly, it allows testers to focus on work that adds the most value, such as creating test scenarios and scrutinizing [29]. There is a small amount of research on automated detection and repair tools for failed testing. However, there is research in this area. These studies have shown that automated test failure detection and healing tools can be effective in identifying and fixing defects in software. Samad et al. [30] discussed details about Self-healing capabilities that automatically identify and heal broken locators during test execution. This eliminates the need for manual intervention and ensures that tests continue to run successfully even after page changes.

Machine learning-powered locator strategies leverage machine learning algorithms to analyze page changes and identify alternative locator strategies that are more robust to updates [31]. This ensures that tests are not dependent on fragile locators that are likely to break. Integrates with existing Selenium test suites, making it easy to adopt without disrupting existing workflows, and supports a wide range of testing frameworks, including Java, Python, JavaScript, and C# [32].

Significantly decreases the time and effort required to maintain automated tests, freeing up testers to concentrate on more valuable tasks. Testers can invest more time in developing new test cases and expanding test coverage, leading to a more comprehensive testing strategy [33]. Streamlines the test automation process, enabling testers to execute tests more efficiently and identify potential issues early in the development cycle. The main focus of this article is to provide a relative analysis of mercantile and open-source web-automated self-healing tools [34].

A feasibility study on the most frequently used tools and a comparison of open sources and commercial ones to determine usability and effectiveness are provided in this paper. The article presents an extensive survey on automated test failure detection and healing tools.This paper aims to find out how different emerging tools and methodologies can be found to assess software system quality constraints in a developed product.

A comparison of various tools based on existing literature, as well as a relative study on different automated testing methodologies, will be conducted, which will help in selecting testing tools and methodologies concerning Cost and Time [35]. To summarize, the emergence of self-healing frameworks in web-based automation mirrors the need to tackle the issue of test maintenance amidst the swiftly changing landscape of web applications [36].

Integrating AI and machine learning technologies opens up new avenues for improving the effectiveness and flexibility of test automation, consequently bolstering the dependability and excellence of web-based software products.

3. Comparative Analysis of Tools

The study concentrated on six tools for application testing automation; their Characteristics, advantages, and disadvantages of using these tools are presented below.

For this paper, automated test failure detection and healing tools chosen are the following:

Mabl [37] is an Artificial Intelligence-based testing automation tool that comes with machine-learning algorithms for detecting and troubleshooting test failures. Auto-healing tests, self-healing test suites, finding root causes, it does it all. Everything has to do with the running of the tests, which is infinitely scalable in a cloud-managed infrastructure, which means all the tests run in parallel. Mabl uses machine learning to identify threats or issues and improve test execution. It identifies issues and triggers alerts of possible impacts for test monitoring; it identifies and eliminates flakiness automatically by detecting changes for web elements and then dynamically updates all related tests. Continuous comparison of test results with the test history allows failures, changes, and regressions to be more easily detected and delivers updated releases more often.

It is again one of the testing frameworks identified in analyzed documents aimed at finding powerful Web elements that have a similar counterpart in their neighborhood, such as in a table using complex XPath selectors for finding elements. One may need to generate the locator for an element by its location, for example, in a list or table. Mabl will track page elements and display what might actually anchor to the target element, along with confidence ratings. It then enables editing of the list of attributes found based on the use case. If broken, it will outline which locator is broken, along with a few possible fixes that a tester has to manually validate. This means that the picked repair will automatically propagate to all tests using that locator.

Appvance IQ [38] is an AI-powered automation platform that provides real self-healing capabilities. It automatically detects and fixes problems with test scripts so they can still work as they should, even when you change the application. Appvance IQ has basic methods of test generation, which include a visual script writer that allows the testers to generate scripts without coding and an Artificial intelligence-independent scripting generator that uses Machine Learning and intelligent generation to create scripts based on actual user tasks. It creates and runs test cases automatically for different platforms and environments.

It examines user actions in order to produce test cases that cover important scenarios encountered by multiple users. The tool detects bugs and vulnerabilities. In addition, it gives

opinions and suggestions for enhancing software quality. Additionally, it displays a dashboard for simple tracking and managing defects. Appvance IQ simplifies defect tracking and management by providing a consolidated defect-tracking dashboard coupled with detailed reports and analytics on testing activities. Appvance IQ enables testers' maximum efficiency in testing their programs by covering most of the test coverage while ensuring the fewest tests are written using this tool, saving time that would be wasted in writing many test cases one by one, compiling them into numerous smaller ones or higher level ways such as applications or modules.

Tricentis Tosca [39] can aid in identifying and resolving test failures. It comprises an integrated defect tracking system that supports troubleshooting the main reason for test failures and making remedies. It allows the creation and running of test cases on various platforms. It leverages state-of-the-art machine learning algorithms to analyze user behavior and generates test scenarios covering edge/corner cases as well as frequent workflows.

It is able to classify errors and vulnerabilities and align them with standards to help increase software quality like the way Tricentis Tosca does. The tool includes an easy-to-read dashboard for monitoring defects and management and provides comprehensive reports on TSTNG activities across platforms or environments.

Applitools [40] helps automate visual UI testing and monitoring with end-to-end tools that are AI-enabled. There's a seamless integration of existing tests during test generation; no manual writing is required. Its Visual AI feature is where machines imitate behavior to identify functions in a web page. Next, its AI analyzes the screen of the application to find UI bugs, e.g., underlying, invisible, off-page, unexpected appearing features. Features are the comparison algorithm that identifies whether the changes are meaningful or just incidents. The algorithm is fully adaptive, so there is no manual configuration and no need for predefined settings.

Applitools has been able to add AI to the validation process by separating test interaction and test validation, helping verify the state of the entire UI in a given scenario, and mapping and measuring thousands of elements, text, and images nearly instantly. You can replace traditional test assertions with a single checkpoint powered by AI that is capable of validating thousands of elements in just seconds using Visual AI. With these smart assertions, the entire interface can be validated in a single command. One simply needs to steer toward the interface that you would like to validate and then organize every element with algorithms to detect any meaningful changes in the future.

Rapise [41] is a self-correcting automated testing program. It uses pattern recognition technology to monitor UI changes in applications, effectively adjusting its testing scripts along the way. For every recorded element, Rapise creates a "full-path locator." A full-path locator includes information on all the attributes of an element as well as its ancestors' attributes in the DOM tree. At the moment of running a recorded test, the element is discovered using a standard XPath locator, and there is no reason to apply this new locator called a "full-path locator". But in case the generated standard XPath during recording or being created manually by a tester for an element does not pass validity checks, a new full-path locator is employed to locate the most appropriate element on the page. There is a powerful and simple visual language in Rapise called Rapise Visual Language (RVL) that allows you to write automated tests without any programming experience.

With this approach, scripts are written without code, a feature that makes automatic testing accessible to domain users and testers who can be far from being programmers. While running automated tests, it's nearly always necessary to verify that the system is functioning correctly and that the data shown on the screen corresponds to assumptions. In addition, there is a strong validation framework that Rapise supports, which can be further developed through scripting in IDE as well as an efficient verification system for creating or including new checking points and validation steps while recording. By using Rapise, you can carry out both positive and negative testing with one script.

Leapwork [42] is a self-healing no-code automation tool. This makes automatic resilient testing possible by enabling modifications of test cases as UI alters in applications. The no-code interface lowers the barrier to entry, allowing business users, QA testers, and developers to create and maintain automated tests. Rapid creation of automated tests without the need for extensive coding knowledge accelerates the automation process. Uses AI to recognize and interact with UI elements, making the tests more robust against UI changes. Automatically adjusts test scripts in response to minor changes in the application under test.

4. Comparative Analysis

With the ever-changing state of software development, choosing test tools correctly becomes a preeminent necessity for both quality and efficiency. This means that every organization should have a better judgment on different parameters or at least understand them. Therefore, this comparative analysis focused attention on some major testing tools that could be compared against specified criteria, such as AI-powered test automation, Self-Healing Support, Programming knowledge, Supported testing types, Reliability, Customization, Performance, Documentation, Pricing, Test case generation. Table 1 shows a comparative analysis of selected testing tools based on evaluation parameters. Figure 1 shows a detailed feature-by-feature comparison of test automation tools with different capabilities and performance metrics.

Table 1. Comparative analysis of the characteristics of automated test failure detection and healing tools

Criteria	Mabl [37]	Appvance IQ [38]	Tricentis Tosca [39]	Applitools [40]	Rapise [41]	Leapwork [42]
AI-powered test automation	Yes	Yes	Yes	Yes	Yes	Yes
Self-Healing Support	Yes	Yes	Yes	No	Yes	Yes
Programming Knowledge	No	Yes	Yes	Yes	Yes	No
Supported Testing Types	UI, API, Mobile	UI, API, Mobile	UI, API, Mobile	UI, API, Mobile	UI, API, Mobile	UI, API, Mobile
Reliability	High	Medium	High	Medium	Medium	High
Customization	Limit	High	High	Limited	High	Moderate
Performance	Fast	Moderate	Fast	Fast	Moderate	Fast
Documentation	Yes	Yes	Yes	Yes	Yes	Yes
Pricing	Pay-per-use	Per-user subscription	Per-user subscription	Per-user subscription	Per-user subscription	Per-user subscription
Test Case Generation	Yes	Yes	Yes	Yes	Yes	Yes

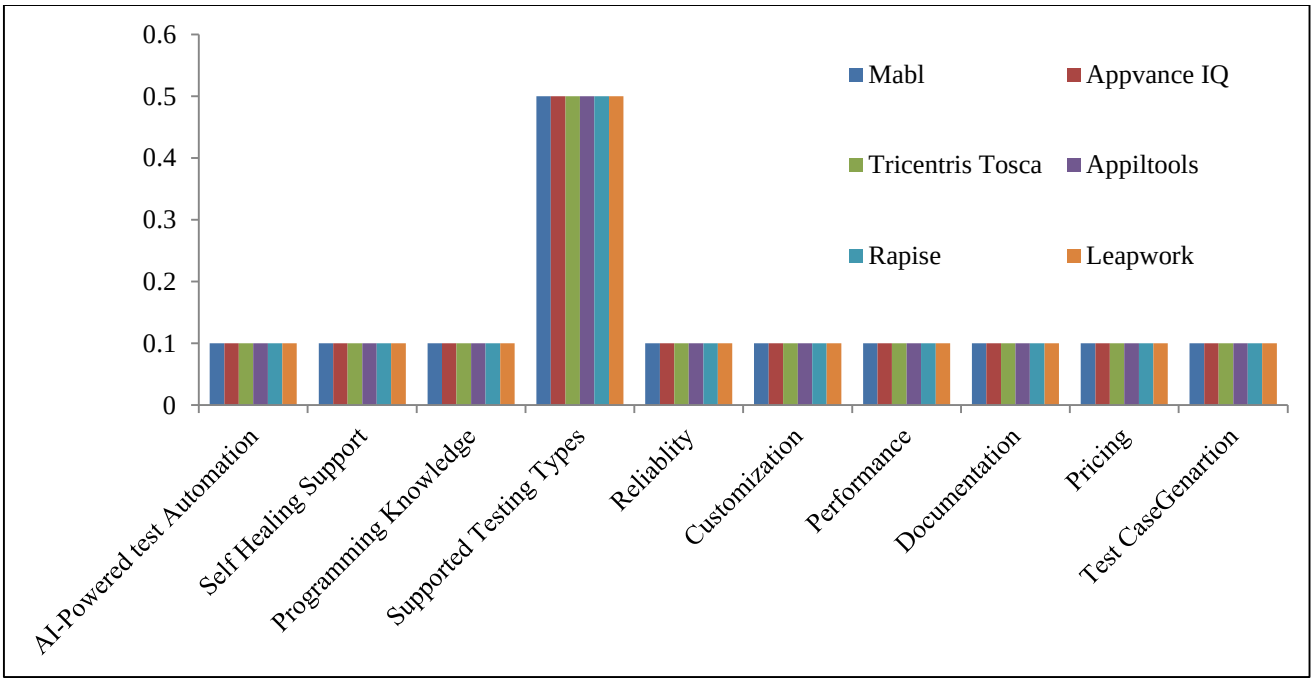

Fig. 1 Comparison of test automation tools

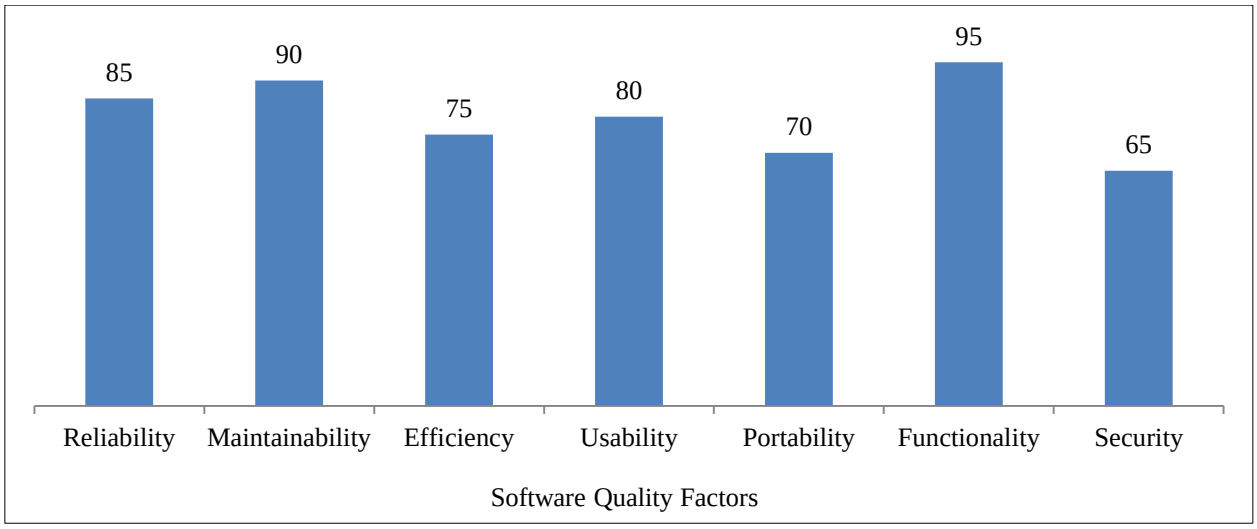

Fig. 2 Quality factors achieved through frequency using automated testing tools

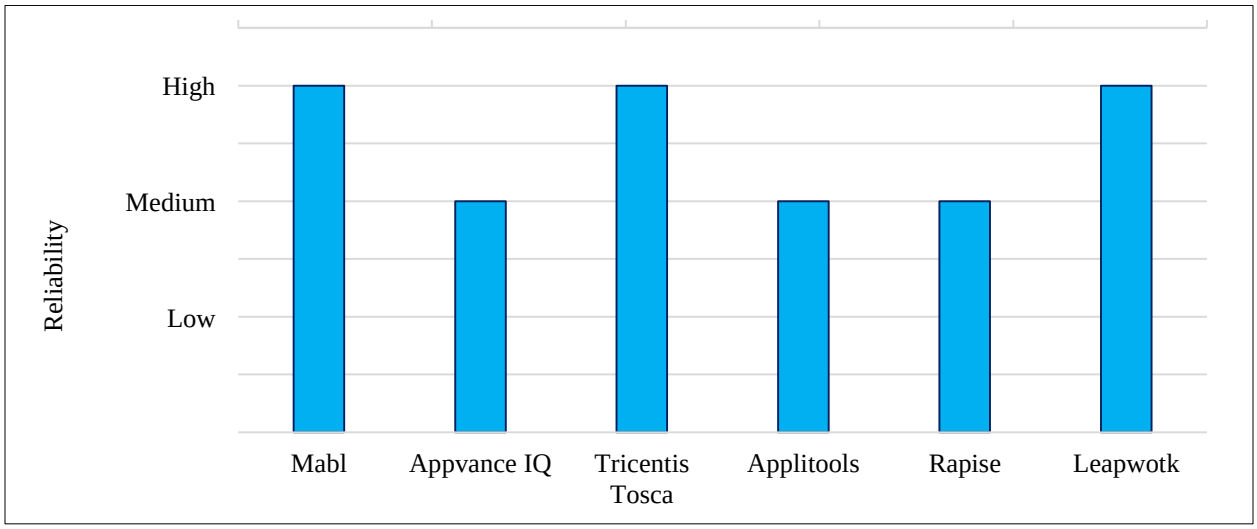

Fig. 3 Reliability of testing tools

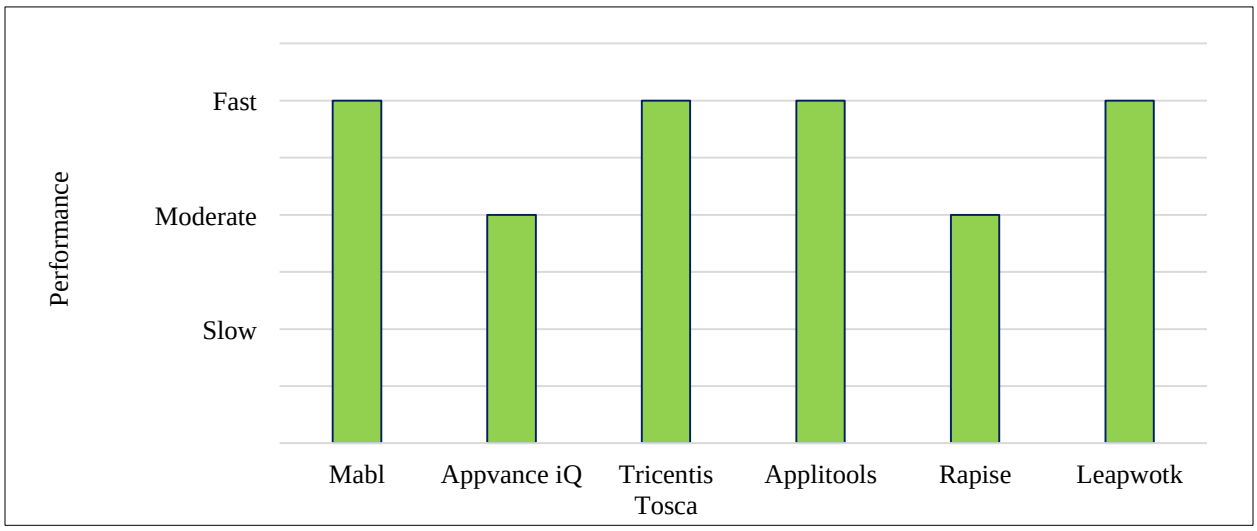

Fig. 4 Performance of testing tools

5. Conclusion

With the increasing shift toward automation across various fields, Test Automation is an advanced alternative for testing groups. Everyone understands AI's potential to re-imagine and re-engineer software development and testing. However, while new automation tools and testing frameworks appear, there is a substantial lack of understanding of AI-based test automation scope, problem-solving strength, and solutions and tools available in this field. In particular, self-repairing web-based frameworks are thoroughly analyzed for their potential and limitations in software testing and test automation. AI, machine learning, and advanced algorithms energy such frameworks. They represent optimistic answers and solutions to a long-standing problem of rewriting test scripts to match the vibrant displays of web applications [43].

5.1. Future Work

The ideal tool is considered easy to install and implement, open source, and freely available. Thus, this tool helps us in further initiative by conducting experiments on these tools and further improving. These will evolve over time to become predictive, basing their anticipations of failure on historical data in order to optimize test coverage. Emerging technologies such as blockchain and IoT combined with self-healing, robust, adaptive, and future-ready testing frameworks are being developed for organizations. The research focuses on improving the quality and reliability of automated test failure identification mechanisms, reducing the time for false negatives and positives, and increasing self-recovery mechanisms. Finally, the paper provides a broad overview of the challenges and the possible routes to address them [44].

References

[1] Itti Hooda, and Rajender Singh Chhillar, "Software Test Process, Testing Types and Techniques," *International Journal of Computer Applications*, vol. 111, no. 13, 2015. [Google Scholar] [Publisher Link]

[2] Glenford J. Myers, Corey Sandler, and Tom Badgett, *The Art of Software Testing*, John Wiley & Sons, 2011. [Google Scholar] [Publisher Link]

[3] Juha Itkonen, Mika V. Mantyla, and Casper Lassenius, "How Do Testers Do It? An Exploratory Study on Manual Testing Practices," *3rd International Symposium on Empirical Software Engineering and Measurement*, Lake Buena Vista, FL, USA, 2009. [CrossRef] [Google Scholar] [Publisher Link]

[4] Suresh Thummalapenta et al., "Automating Test Automation," *34th International Conference on Software Engineering (ICSE)*, Zurich, Switzerland, 2012. [CrossRef] [Google Scholar] [Publisher Link]

[5] Shikha, and Kailash Bahl, "Software Testing Tools & Techniques for Web Applications," *International Journal of Engineering and Technical Research (IJETR)*, vol. 3, no. 5, 2015. [Google Scholar] [Publisher Link]

[6] Nazia Islam, "A Comparative Study of Automated Software Testing Tools," *Culminating Projects in Computer Science and Information Technology*, 2016. [Google Scholar] [Publisher Link]

[7] Vishawjyoti, and Sachin Sharma, "Study and Analysis of Automation Testing Techniques," *Journal of Global Research in Computer Science*, vol. 3, no. 12, pp. 36-43, 2012. [Publisher Link]

[8] Andreas Spillner, and Tilo Linz, *Software Testing Foundations: A Study Guide for the Certified Tester Exam-Foundation Level-ISTQB® Compliant*, dpunkt. Verlag, 5th ed., 2021. [Google Scholar] [Publisher Link]

[9] Dorothy Graham, and Mark Fewster, "Experiences of Test Automation: Case Studies of Software Test Automation," *Addison-Wesley Professional*, 2012. [Google Scholar] [Publisher Link]

[10] Mubarak Albarka Umar, and Chen Zhanfang, "A Study of Automated Software Testing: Automation Tools and Frameworks," *International Journal of Computer Science Engineering (IJCSE)*, vol. 8, no. 6, pp. 217-225, 2019. [Google Scholar] [Publisher Link]

[11] Anand Singh Gadwal, and Lalji Prasad, "Comparative Review of the Literature of Automated Testing Tools," 2020. [Google Scholar]

[12] Soorajit Mukherjee, "Self-Healing Test Automation," *Medium*, 2020. [Publisher Link]

[13] Gemma Catolino et al., "Not All Bugs are the Same: Understanding, Characterizing, and Classifying Bug Types," *Journal of Systems and Software*, vol. 152, pp. 165-181, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[14] Dhaya Sindhu Battina, "Artificial Intelligence in Software Test Automation: A Systematic Literature Review," *International Journal of Emerging Technologies and Innovative Research*, vol. 6, no. 12, 2019. [Google Scholar] [Publisher Link]

[15] Shahrokh Jalilian, and Shafagat J. Mahmudova. "Automatic Generation of Test Cases for Error Detection Using the Extended Imperialist Competitive Algorithm," *Problems of Information Society*, vol. 13, no. 2, pp. 46-54, 2022. [Google Scholar] [Publisher Link]

[16] Sutharsan Chiranjeevi Partha Saarathy, Suresh Bathrachalam, and Bharath Kumar Rajendran, "Self-Healing Test Automation Framework Using AI and ML," *International Journal of Strategic Management*, vol. 3, no. 3, pp. 45-77, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[17] Shahbaa I. Khalee, and Raghda Anan, "A Review Paper: Optimal Test Cases for Regression Testing Using Artificial Intelligent Techniques," *International Journal of Electrical & Computer* Engineering, vol. 13, no. 2, pp. 1803-1816 2023. [CrossRef] [Google Scholar] [Publisher Link]

[18] Gabaire Elmi Bile, "The Utilization of Log Files Generated by Test Executions: A Systematic Literature Review," *Digitala Vetenskapliga Arkivet*, 2023. [Google Scholar] [Publisher Link]

[19] Szymon Stradowski, and Lech Madeyski, "Machine Learning in Software Defect Prediction: A Business-Driven Systematic Mapping Study," *Information and Software Technology*, vol. 155, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[20] Shravan Pargaonkar, "Advancements in Security Testing: A Comprehensive Review of Methodologies and Emerging Trends in Software Quality Engineering," *International Journal of Science and Research (IJSR)*, vol. 12, no. 9, pp. 61-66, 2023. [Google Scholar] [Publisher Link]

[21] Sabina-Cristiana Necula, Florin Dumitriu, and Valerică Greavu-Șerban, "A Systematic Literature Review on using Natural Language Processing in Software Requirements Engineering," *Electronics*, vol. 13, no. 11, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[22] Zahra Yazdanparast, "A Survey on Self-Healing Software System," *arXiv*, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[23] Yasunari Matsuzaka, and Ryu Yashiro, "AI-Based Computer Vision Techniques and Expert Systems," *AI*, vol. 4, no. 1, pp. 289-302, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[24] Gagan Kumar, Vinay Chopra, and Dinesh Gupta, "Systematic Literature Review in Software Test Data Generation," *Emerging Trends in Engineering and Management*, pp. 91-107. 2023. [CrossRef] [Google Scholar] [Publisher Link]

[25] C. Anjali, Julia Punitha Malar Dhas, and J. Amar Pratap Singh, "Automated Program and Software Defect Root Cause Analysis using Machine Learning Techniques," *Automation: Journal of Automation, Measurement, Electronics, Computing and Communications*, vol. 64, no. 4, pp. 878-885, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[26] Amin Milani Fard, Mehdi Mirzaaghaei, and Ali Mesbah, "Leveraging Existing Tests in Automated Test Generation for Web Applications," *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, pp. 67-78, 2014. [CrossRef] [Google Scholar] [Publisher Link]

[27] Shivkumar Goel, and Kshitija Vartak, "Selenium with Support of both Test NG and Cucumber Frameworks," *International Journal of Computer Applications*, vol. 180, no. 51, 2018. [Publisher Link]

[28] Ghada Alsuwailem, and Ohoud Alharbi, "Utilizing Machine Learning for Predicting Software Faults through Selenium Testing Tool," *International Journal of Computations, Information and Manufacturing (IJCIM)*, vol. 3, no. 2, 13-27, 2023. [Google Scholar] [Publisher Link]

[29] Dalia Alamleh, "Utilizing AI in Test Automation to Perform Functional Testing on Web Application," *Science and Information Conference*, Springe, vol. 507, pp. 359-377, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[30] Abdus Samad et al., "A Cognitive Approach in Software Automation Testing," *Proceedings of the International Conference on Innovative Computing & Communication*, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[31] Hadeel Mohamed Eladawy, Amr E. Mohamed, and Sameh A. Salem, "A New Algorithm for Repairing Web-Locators Using Optimization Techniques," *13th International Conference on Computer Engineering and Systems, Cairo, Egypt*, pp. 327-331, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[32] Harshita Wardhan, and Suman Madan, "Study on Functioning of Selenium Testing Tool," *International Research Journal of Modernization in Engineering Technology and Science*, vol. 3, no. 4, 2021. [Google Scholar] [Publisher Link]

[33] Mubarak Albarka Umar, "A Study of Software Testing: Categories, Levels, Techniques, and Types," *Authorea Preprints*, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[34] Filippo Ricca, Alessandro Marchetto, and Andrea Stocco, "AI-Based Test Automation: A Grey Literature Analysis," *IEEE International Conference on Software Testing, Verification and Validation Workshops*, Porto de Galinhas, Brazil, pp. 263-270, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[35] Rohit Khankhoje, "Effortless Test Maintenance: A Critical Review of Self-Healing Frameworks," *International Journal for Research in Applied Science and Engineering Technology*, vol. 11, no. 10, 2023. [Google Scholar] [Publisher Link]

[36] João Paulo Magalhães, and Luis Moura Silva, "SHoWA: A Self-Healing Framework for Web-Based Applications," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 10, no. 1, pp. 1-28, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[37] Rohit Khankhoje, "An Intelligent Apitesting: Unleashing the Power of AI," *International Journal of Software Engineering and Application*, vol. 15, no. 1, pp. 1-8, 2024. [Google Scholar] [Publisher Link]

[38] Moez Krichen, "A Survey on Formal Verification and Validation Techniques for the Internet of Things," *Applied Sciences*, vol. 13, no. 14, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[39] Flaviu Fuior, "An Overview of Some Tools for Automated Testing of Software Applications," *Romanian Journal of Information Technology & Automatic Control*, vol. 29, no. 3, pp. 97-106, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[40] Phuoc Pham, Vu Nguyen, and Tien Nguyen, "A Review of AI-Augmented End-to-End Test Automation Tools," *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, pp. 1-4, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[41] Danijel Radošević, Nikola Mrvac, and Andrija Bernik, "Robotic Process Automation with Optoklik," *Preprints*, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[42] Marina Cernat, Adelina Nicoleta Staicu, and Alin Stefanescu, "Towards Automated Testing of RPA Implementations," *Proceedings of the 11th ACM SIGSOFT International Workshop on Automating TEST Case Design, Selection, and Evaluation*, pp. 21-24, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[43] Yuvaraja Devarajan, "A Review on Intelligent Process Automation," *International Journal of Computer Applications*, vol. 182, no. 36, 2019. [Google Scholar] [Publisher Link]

[44] Kam K.H. Ng et al., "A Systematic Literature Review on Intelligent Automation: Aligning Concepts from Theory, Practice, and Future Perspectives," *Advanced Engineering Informatics*, vol. 47, 2021. [CrossRef] [Google Scholar] [Publisher Link]