International Journal of Core Engineering & Management
Volume-7, Issue-09, 2024 ISSN No: 2348-9510

ADVANCEMENTS IN AUTOMATED SOFTWARE TESTING: A COMPREHENSIVE REVIEW OF CURRENT PRACTICES AND TOOLS

Prathyusha Nama
Test Architecture Manager
Align Technology Inc., USA

Abstract

Software testing is an important process which is used in software engineering that helps in determining the quality and functionalities of an application. This review paper aims to outline the development and importance of software testing especially concerning automation testing frameworks. The purpose is to identify and discuss how the evolution of testing patterns and paradigms enhanced the efficacy of the testing process. The paper gives an Introduction to software testing basics, which is aimed at explaining the need for testing of software. It then describes the Evolution of Software Testing pointing out phases where testing has evolved from being a manual process into an automation process. A general understanding of Software Testing can be gathered from the Overview of Software Testing which offers direction to numerous contemporary methods and their uses. Analysis of the primary test automation tools like Selenium, TestNG, Cucumber, JMeter and Cypress features and advantages demonstrate exactly how they enrich testing processes within automation and interactions with other instruments. It is with this rationale that this review provides thorough evaluation, emphasizing on the revolutionary effect of the above innovations in current trends of software testing as well as their implications on the future trends of software quality assurance.

Keywords: Software Testing, Automation, Agile, DevOps, Artificial Intelligence, Machine Learning, Continuous Testing, Test Automation Tools, Integration Testing, System Testing, Automated Test Case Generation..

I. INTRODUCTION

Software testing is a crucial part of developing software since it ensures the final product satisfies all needs and specifications set forth by the user. One approach to software testing is automated testing, while another is manual testing. Software testers find automated testing to be more successful in terms of time, cost, and usability since it allows them to easily automate the software testing process. Open source and commercial options abound when it comes to automated testing tools[1][2]. A functional application is the end result of software development procedures including software programming, documentation, and testing. It is essential to do software testing in order to ensure that the final product satisfies user needs and is fully functioning. The proper execution of every system application relies heavily on it. Manual testing is one option, while software testing tools allow for automation of the process. By simulating actual user activity, testers may ensure that the majority of an application's features work as intended during manual testing. Thus, manual testing is labor-intensive, time-consuming, and error-prone. Consider it a great alternative for smaller businesses without the capital to invest in fully automated systems[3].
The problems with manual testing may be solved by using automated testing. Automated testing enables testers to generate test cases that can be reused and repeated. It is then possible to run these test cases as often as required. The use of automated software testing tools to ensure the

application's functioning and quality is becoming more and more essential as software development processes get more sophisticated. In terms of usability, cost, and time, an automated test performs better. There is a vast array of free source and commercial automated testing solutions on the market. Certain software tools are restricted to a certain kind of language and can only do a particular sort of testing. Software testing solutions that provide more features and capability and support a wider variety of applications may come at an extra expense. A user might choose the appropriate testing tool for their environment by being aware of how one differs from the other[4][5]. The objective of this research is to conduct a comparative analysis of the existing software automated testing tools by contrasting their characteristics with respect to testing type, software support, cost, licensing, and other factors.

1. Research Motivation and Significance/Aim

This research aims to explore and evaluate advancements in automated software testing to address the escalating complexity and rapid development cycles of modern software systems. The motivation behind this study is driven by an inadequacy of traditional manual testing methods in keeping pace with the demands of continuous integration and agile development. Automated testing emerges as a crucial solution, offering faster feedback, enhanced coverage, and improved efficiency. By investigating these advancements, this research aims to demonstrate they can significantly improve software quality and reduce costs, thus aligning with the evolving needs of contemporary software development practices.

2. Research contribution

The paper contributions are as follows:

- Provides a thorough review of current practices and advancements in automated software testing, offering a detailed understanding of the latest trends and technologies in the field.
- Evaluates various automated testing tools and frameworks, highlighting their effectiveness, limitations, and best use cases, thus aiding in the selection of appropriate tools for different testing needs.
- Identifies and analyzes best practices in automated testing, offering valuable insights and guidelines for optimizing testing strategies and enhancing overall testing efficiency.
- Explores how automated testing tools integrate with modern development methodologies, such as continuous integration and DevOps, demonstrating their role in contemporary software development processes.
- Addresses common challenges in automated testing, such as tool limitations and integration issues, and proposes practical solutions to overcome these obstacles.
- Suggests future research directions and potential advancements in automated testing, guiding ongoing and future innovations to further improve testing processes and technologies.

II. EVOLUTION OF SOFTWARE TESTING

Quantitative testing has evolved over the years as one approaches the field of software engineering that focuses on selecting the most efficient way to test software. Testing in the early period of computing was simple and mostly done by hand rather than being aided by computer programs. The process was usually about executing programs and then comparing the obtained results with expected ones, and in general, this work was time-consuming and rather vulnerable to more mistakes made by a human being[6][5].

The testing of software was in its infancy and was not systematic during the 1950s and 1960s. Developers would run concrete test cases and correct faults individually during the development

International Journal of Core Engineering & Management
Volume-7, Issue-09, 2024 ISSN No: 2348-9510

process. This was feasible for small programs but was soon seen to be unworkable as software systems evolved. There was often no rigorous way of testing the software and defects were allowed to creep into the final product and hence, people realized the need for more formalized and systematic testing [7][8].

By the 1970s, software industry began to realize the need for structured testing approaches to be followed. This period was characterized by introduction of such fundamental types of testing like integration testing, unit testing, and system testing. academicians and practitioners began to call for systematic and formalized ways of testing hence shaping of different testing strategies and methods. Another factor that played role in the evolution of testing practices was the introduction of structured programming and programming languages that included error checking capabilities[9][10].

The 1980 s are considered the beginning of automated testing domination in the software development process. When software systems started to develop complexities, the problem of performing manual testing became very conspicuous. Different tools that could support the scripts of tests and provide the execution of many tests simultaneously came to the fore. The automation of early test tools and script languages made this era more effective in the testing processes of software systems as compared to prior eras[11][12].

The later enhancements of the software testing practices came in the 1990's and 2000s through the enhanced automated testing tools. One of the key values of the Agile methodology and the implementation of DevOps was a continuous integration and continuous testing which meant that testing had to be an inherent part of the software developing process. TDD and BDD frameworks also brought into discussion the topic of testing right from the beginning and writing tests based on the users' stories and their needs [13].

More recent, software testing has remained under development through an application of AI as well as ML. The aforesaid technologies have been employed in improving test automation, determining better test cases, and, planning possible faulty areas. Further, the emergence of the cloud model and the advancements in the Microservices architecture and containerization pose several new tests and realities to testing. Some current testing practices are the performance, security, and usability testing so that the body of software will not only operate correctly, but also capable of performing what the user expects with it and can work as a stressed system. The evolution phases of testing are given below in Figure 1.

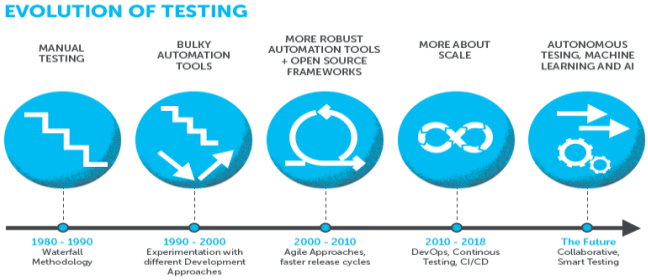

Figure 1: Evolution of Testing

1. Milestones in Automated Testing

The step involved in the automation of testing is important to illustrate the evolution and development of testing tools and procedure with reference to the assurance of software quality.

- The use of automation in the testing process became more popular in the middle of the 1980s because of the growth of complexity in software solutions. Originally, automated testing tools were rudimentary and frequently addressed only certain areas of testing, such as unit or

International Journal of Core Engineering & Management
Volume-7, Issue-09, 2024 ISSN No: 2348-9510

regression testing. These tools assisted in the acceleration of the testing process and in decreasing manual work thus leading to increased automation.

- Advanced automated testing frameworks and tools began emerging in the early 1990s. It was also the time when test management systems were introduced to ease the management and undertaking of test cases. Mercury Interactive's Win Runner and Rationale's Robot were some of the other tools that gained importance with the added benefits like script recording and playback. These tools helped in evolved and creation of reusable test script and enhanced the process of regression testing and became more essential with complex systems.
- The early 2000s were characterized by the growth of open-source testing tools and frameworks. The introduction of tools like JUnit, Selenium, and TestNG revolutionized automated testing by making powerful testing solutions accessible to a broader audience. Selenium, in particular, emerged as a popular choice for web application testing, providing a robust framework for automating browser interactions and validating web application behavior across different platforms.
- Another next major driver has been the introduction of Agile and DevOps methodologies around the 2010s, which cemented automated testing. CI/CD workflows became the norm where automated testing can't be manually integrated into the development workflow. Automation was the main characteristic of the tools placed in the presented Figure 3 As dependencies management tools there were used Jenkins and Travis CI As for the testing tools Cucumber and Spec Flow supported the Behavior Driven Development approach which allowed teams to describe tests basing on business needs.
- The emergence of AI and ML in the last few years have been noted as a major development in the area of automation in testing. Current testing tools include AI that renders the case generation of tests, predicts the occurrence of defects, and improves the coverage of tests. These are some of the developments that assist in solving the difficulties that are experienced when it comes to testing thoroughly integrated complex application like those based on Microservices or the cloud.
- In particular, the achievements and further development of automated testing show the increasing demands for effectiveness, precision, and coverage of testing services. Over time, there will be updates in the tools and the methodologies applied in the automated testing field to meet challenges and enhance the quality of the software.

III. OVERVIEW OF SOFTWARE TESTING

The accuracy of a program can never be known. In order to find problems that lead to failure, testing is conducted. A higher degree of trust in the software's functioning is achieved via a variety of testing procedures. When evaluating software, functional testing involves comparing the system to its intended use cases. In order to determine whether a system is simple to use, usability testing takes the user's viewpoint into account. A subset of software testing known as "security testing" checks for potential weak points in a system and takes other precautions to make sure that sensitive information and hardware are protected from intrusion. The purpose of performance testing, a subset of software testing, is to determine the system's stability and responsiveness under a defined load. The goal of regression testing, a subset of software testing, is to ensure that the program's behavior has remained unchanged after implementing modifications such as improvements or bug fixes. It is possible to determine if a system satisfies both internal and external standards via a process known as compliance testing [14]. Various more tailored tests are carried out to guarantee the program's operational stability, depending on the kind of software.

International Journal of Core Engineering & Management
Volume-7, Issue-09, 2024 ISSN No: 2348-9510

You may generally categorise software testing as follows:

- **Static Testing**: Understanding and analysing the program code is the first step in creating static tests. Symbolic execution methods are then used to simulate abstract program executions, simulating program executions and computing inputs to drive along predefined branching or execution routes without doing the program itself[15].
- **Dynamic Testing**: In a DART-directed search, every new input vector aims to direct the program's execution in a different direction. By repeating the process, a directed search like this attempts to force the program to go through every potential execution path[16].

Table 1: Advantages and Disadvantages of Software Testing

Advantages	Disadvantages
Increases the speed at which bugs may be found with greater accuracy	Inherent knowledge of the tool required
The procedure is quite efficient, which saves both time and energy.	Choosing the appropriate strategy and tool takes a lot of time
The test script can be comfortably repeated	The initial investment in a test tool is substantial
Enhances software correctness and the quality of the testing procedure	Maintaining tests is a significant task if the playback strategy is used
Enhanced coverage due to the use of many testing instruments for concurrent testing instances	Test maintenance is more demanding if the playback strategy is used.

1. Automated Testing

The term "software testing automation" refers to the practice of using an external automation tool to mimic the actions of a human tester by creating a program in a programming or scripting language. The process includes developing toolkits for testing the implemented source code. More automation of the testing procedures is its intended outcome. Program development and test script writing are both examples of development tasks; program development pertains to the application itself, while test script writing pertains to the scripts that will evaluate the application. The process of automation is shown in Figure 2.

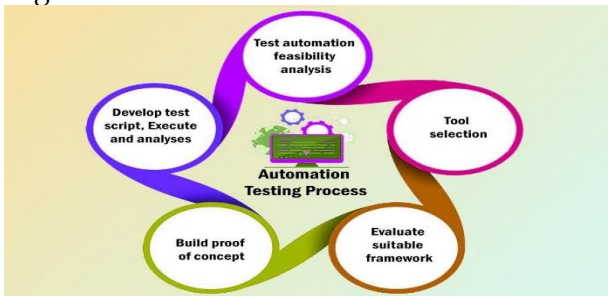

Figure 2: Process of automation testing process.

International Journal of Core Engineering & Management
Volume-7, Issue-09, 2024 ISSN No: 2348-9510

Tests that can be automated:

- The system's hard-to-reach regions include things like background processes, file logging, and database entries.
- Features that are used often yet have a significant potential for mistakes, such as payment systems and registrations. Automation guarantees rapid mistakes since critical functional tests often take several minutes.
- Load testing ensure that a system can handle a large number of requests without crashing.
- Data searches, multi-field form input, and preservation verification are all examples of template activities.
- Validation messages: The validity should be verified, and erroneous data should be entered in the fields.
- Situations that take a long time from beginning to conclusion.
- Accurate mathematical calculations, including data verification, for applications like analytical processing and accounting. 8. Making sure the data search was accurate (Korel, 1990).

2. Cost of Testing Process

Choosing the right testing methodologies and automated testing tools, reducing the number of test cases, prioritizing test cases, etc., may all help bring down the cost of testing. Test automation streamlines the process by running the same test cases over and over again, which saves time and effort for both developers and testers and allows for faster bug detection. Enhancing the accuracy of testing while reducing the possibility of errors caused by human error is another benefit. The market offers a wide range of testing tool kinds. With the right testing tools, almost any kind of software testing work may be automatically completed. The industry uses a wide variety of software testing technologies these days[17].

- **Test Management tools:** These kinds of tools are employed in the management of test cases, automated test case execution, planning, etc.
- **Unit Testing**: The individual modules are tested using unit testing techniques to make sure they function as expected.
- **Integration Testing Tools:** When many modules are linked together, issues are found using integration testing methods.
- **Regression Testing Tools**: These tools are employed to ensure that the updated software component does not impact the functionality of the remaining program.
- **Performance Testing Tools**: The applications' performance is tested using these sorts of techniques under different loads.
- **Bug Tracking Tools:** During the testing stage, defects are found using bug tracking tools.
- **Cross-Browser Testing Tools**: To ensure the app works properly across a wide range of devices and web browsers, developers use these testing methods.
- **Security Testing Tools:** The purpose of security testing tools is to identify software flaws that might be used by hostile individuals.
- **UI Testing tools:** These testing tools guarantee that the program is easy to use [18].

3. Effectiveness of testing process

Additionally, it is dependent on the software testing instruments used in the automated tests. Each testing instrument has benefits and drawbacks. Choosing the right instrument is a really difficult process. When the wrong testing tools are used, high-quality software cannot be produced in the allotted time and budget. The following criteria should be taken into consideration while choosing tools

International Journal of Core Engineering & Management
Volume-7, Issue-09, 2024 ISSN No: 2348-9510

- Skills of the team to best utilize the tool
- Budget
- Required features
- Script maintenance and reusability
- Integration capabilities
- Technical support

IV. FRAMEWORKS FOR TEST AUTOMATION

Automated frameworks make automated testing easier for testers by streamlining the process of developing and running tests. A standard automation framework offers a setting where test plans can be carried out and consistent results may be produced. These are specialized tools that help you with routine activities related to test automation. With the help of a test runner, action recording tool, or web testing tool, you may spend less time creating test scripts and more time checking for quality. Software development may be enhanced by the use of test automation, a tried-and-true method that keeps costs down. Test outcomes and QA timelines are therefore greatly affected by the test automation framework you choose. Automating software testing requires careful protocol integration. The concept of a test framework in automation refers to the scalable and coordinated framework that supports the design, fabrication, and implementation of a test automation suite[19]. The framework of Test automation are given Figure 3.

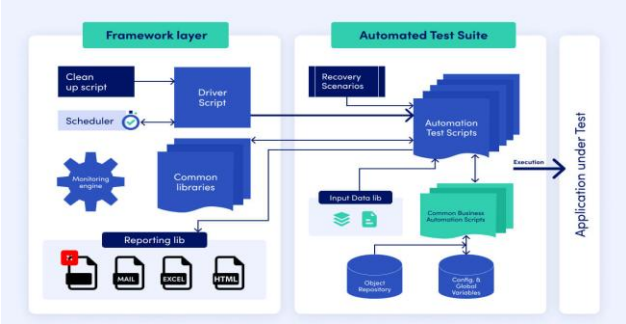

Figure 3: Frame work of Test automation

Software execution efficiency and reputation may be shaped with the help of the framework's many tools and techniques. Code standards, object repositories, storage for test findings, and details on how to access external resources are all part of the physical structures used to verify the construction and logical interaction of the basic components. A framework guarantees a consistent method for adding, editing, and excluding test scripts and functions. The architecture guarantees scalability and ultimate dependability throughout execution while minimizing energy usage[20].

1. Types of Frameworks for Software Testing

Test automation frameworks may be broadly classified, and each subcategory has its own set of requirements for supporting infrastructure. Their degree of automation, important features like reusability, and ease of repair and maintenance are typical points of differentiation. Software experts and testing teams must always priorities making the correct choice of automation framework. The following are descriptions of a few test and analysis frameworks that are common in automated testing [21][22]. The key benefits of automation framework are given in Figure 4 and their types are given in Figure 5.

International Journal of Core Engineering & Management
Volume-7, Issue-09, 2024 ISSN No: 2348-9510

Figure 4: key benefits of the test Automation framework

- **Linear Scripting framework:** For every test case, a new test script is created and run separately. After recording each test step—including navigation, browsing, user inputs, and checkpoint enforcement—testers run the scripts to execute the tests. Applications with modest requirements often use this framework type.
- **Data-Driven Automation:** The data-driven approach, in contrast to the modular structure, may simply pass and ensure that the test scripts function correctly for various data sets. The use of this framework allows for the reduction of test scripts while still providing test coverage via the use of reusable tests. Coding abilities and knowledge of test automation are necessary for the design of this framework.
- **Keyword Driven Automation**: Automation test scripts are executed just by referring to the keywords included in the Excel sheet; this method is also called action word-based test automation. Using keywords, the tester may create a script for any kind of test automation. This model is comparable to data-driven testing. Despite the fact that designing keyword-driven automation might be a time-consuming process, anybody can develop test scripts. However, workers who are proficient in test automation are still required.
- **Modular Automation:** The whole program is divided into more manageable, standalone modules, each of which serves as the basis for a separate test script module created by testers. To create bigger test scripts that accomplish the necessary scenario, these scripts may be altered and merged.
- **Hybrid Automation**: Hybrid test automation integrates two or more frameworks, as the name implies. In the present market, a number of teams are employing this framework in an effort to capitalize on the advantages of other frameworks.
- **Behavior-Driven:** This framework's development goal is to provide a multi-purpose platform that enables ongoing participation by business analysts, developers, testers, and others.

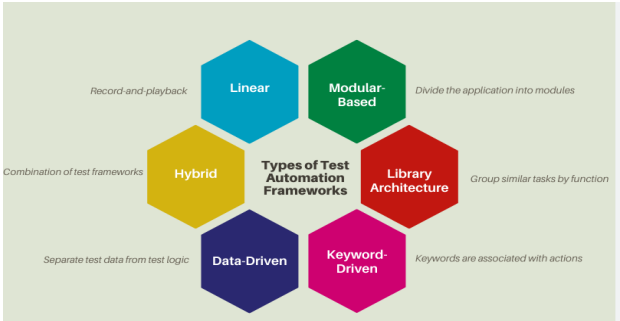

Figure 5: Types of automation frameworks

International Journal of Core Engineering & Management
Volume-7, Issue-09, 2024 ISSN No: 2348-9510

2. The Significance of Test Automation Framework

The Importance of Test Automation Frameworks An automation testing framework is a platform that is built using a qualifying set of assumptions and integrates multiple hardware and software resources. It also uses various tools for automation testing and web service automation. Automated test scripts may be efficiently designed and developed using this framework, and problems or faults in the AUT can be reliably analyzed. When it comes to automating IT and related software processes, the test automation framework is crucial. Using the automation protocols, testers may record tests and install an integrated script. The testing team can do more with a comprehensive automation framework, including making test components more reusable and useful, creating scripts that are easier to maintain, and having access to high-end automation scripts[23]. Efficiency, speed, increased test accuracy, less disruptive maintenance costs, and reduced hazards are all benefits of incorporating a framework into the testing process. Many software tests rely on these frameworks as dependable building blocks, including Functional Testing, Unit Testing, and Performance testing. Software testing automation frameworks serve a variety of vital purposes, including the efficient identification of objects, their organization for future reuse in test scripts, the execution of actions on these objects, and their evaluation to produce the desired outcomes[24].

V. OVERVIEW OF POPULAR FRAMEWORKS OF TESTING

The purpose of software testing, a fundamental activity in software engineering, is to ensure that the produced program performs as planned. Its goal is to guarantee that the software system is bug-free. In order to evaluate different desired qualities, testing entails running software or system components. Performing a thorough analysis of software to see whether it satisfies the set criteria and to locate and fix any bugs or mistakes is known as software testing.

1. **Selenium** uses the Page Object Model (POM). POM is a popular design pattern in Selenium testing. By doing so, it isolates the automation code from the web application's user interface component testing. The Selenium framework is very compatible with other programs. This includes technologies for continuous delivery (CD) and continuous integration (CI). This feature enables automated testing, which may aid in finding and correcting bugs early on in the development cycle, ultimately leading to software that is easier to maintain.

2. **The TestNG framework** allows developers to organize their tests into groups, suites, and dependencies. Large test suites are now easy to organize and maintain. Developers can find and solve issues quickly when tests are organized. Thus, it is reasonable to assume that the organized testing feature contributes to making software more maintainable. With TestNG, test-driven development is made possible. Developers may use this functionality to create automated tests prior to coding. The TestNG framework allows the testers to run the test cases again to see whether the defects have been addressed. This benefit allows testers to cut down on the time and energy needed to resolve issues. Continuous integration is another feature that TestNG offers.

3. The **Cucumber framework** encourages teamwork across stakeholders, including developers and testers. This framework may be used by stakeholders and developers to create and execute tests. Software design and maintainability may be enhanced when developers work together to clarify software requirements and communicate more effectively. Testers may now construct test cases in a structured, human-readable fashion using Gherkin syntax, which is just one

more way the Cucumber framework improves maintainability. The Cucumber framework offers a distinct division of responsibilities. Make use of this benefit by keeping the development and test codes apart. Instead of attempting to navigate a convoluted, monolithic structure, developers may concentrate on the relevant sections of the code when modifications are needed [25].

4. **JMeter** helps to identify performance bottlenecks. JMeter allows developers to simulate various scenarios in order to identify bottlenecks in program performance. By identifying these issues, developers may improve the code's optimization and program maintainability. With JMeter, testers may write scripts for reusable tests. The software's many components may be tested using these reusable test scripts. Reusability streamlines the testing process and saves time. Developers may easily update and alter tests as required by using reusable scripts. Test automation is encouraged by JMeter. By automating test execution using JMeter, testing time and effort may be significantly reduced.

5. **Cypress** gives real-time feedback on test outcomes and application behaviour when running test scripts. This aids in the testers' ability to identify and address problems. By guaranteeing that problems are resolved as soon as they are discovered, this feature enhances the maintainability of software. Cypress furthermore offers a precise and comprehensive error message. Testers can rapidly determine the source of problems with the help of this error message. The Cypress framework facilitates testing process automation by integrating with CI/CD pipelines. Testers may find problems earlier in the development process by automating the testing process, which makes fixing them simpler and less expensive. Cypress gives developers the ability to rewind time and see precisely what transpired during a particular test run.

VI. LITERATURE OF REVIEW

The literature on developments in software testing is thoroughly reviewed in this section. The below Table 2 outlines the objectives, accomplishments, challenges, and proposed future work of the referenced papers, giving a clear picture of the advancements in software testing methodologies.

In [26], the whole Agile software testing process is included in the report. Testing is now an essential part of software development alongside coding, as agile development has shown, rather than being a separate step. To create software solutions of the greatest caliber, the agile team uses a comprehensive approach. Teams may determine and arrange the required tests with the help of the categorization offered by its quadrants. This article aims to explore different and enhanced Agile Testing Processes and approaches in order to enhance quality assurance. Furthermore, discussed is the software testing methodology used in Agile development. Testing methods may be added to agile development processes to make them better and more beneficial.

In [27], This study aims to elucidate the fundamental principles of AI and its potential applications in Software Testing. The results show that using AI applications produces substantially better software testing outcomes. Artificial intelligence-driven testing will improve quality assurance in the future. The organization will be more efficient in producing more advanced software for the market as AI-based software testing would save time. Intelligent software testing for complicated software applications will be made possible by the technique of integrating artificial intelligence into software testing.

International Journal of Core Engineering & Management
Volume-7, Issue-09, 2024 ISSN No: 2348-9510

In [28], framework increases the precision and efficacy of performance, scalability, and security testing by using emulation and simulation approaches to mimic various deployment situations, network circumstances, and user interactions. Comparing experimental outcomes to conventional approaches, testing efficacy, accuracy, and efficiency have significantly improved. To provide just one example, the distributed testing framework cut testing time by 18.75% and decreased CPU and memory utilization. To guarantee a smooth user experience and drive the continuing evolution of cloud technologies, these solutions provide the groundwork for more powerful and dependable cloud-based software.

In [29], Introducing Evo Crash, an innovative method for automatic crash reproduction that relies on a new evolutionary algorithm known as Guided Genetic Algorithm (GGA). They detail our controlled experiment that evaluated the effectiveness of Evo Crash tests in debugging and our empirical investigation that used Evo Crash to simulate 54 real-world crashes. In light of our findings. They also noticed that, in comparison to developers debugging and correcting code without utilizing Evo Crash tests, developers who use Evo Crash are able to submit patches more often and in less time.

In [30], provide our machine learning-based method, which takes into account both a special metric known as the "Component Dependency score" and parameters from System Testing matrices in order to identify possible fault regions. Additionally, this technique aids in managing the software quality of constantly changing software.

In [31], describes, evaluates, and summarizes the difficulties that come with developing software that is ML-enabled rather than standard software development. However, by using the SE approach to engineer the creation of ML-enabled software, the research was able to uncover advancements for ML-enabled software, such as the automation of mismatch detection, which arises because of the many stakeholder views. The stakeholders—SE and ML in particular—need to work together, set aside their differences, and acquire further expertise and knowledge in order to determine user needs. This may be done via education, training, and collaboration. Ultimately, in order to create the ML software development process, the study reframed the conventional SE development method.

In [32] , Collaborative bug finding aids in the discovery of seventeen more problems by rookie Android app testers in the first two settings of a software testing course. The students acknowledge that finding pertinent bug reports could be a time-consuming task. In response, they propose Bugine, a method that suggests appropriate GitHub issues for a specific app automatically. Based on our findings, Bugine can identify 34 new bugs. Collaborative bug hunting yields 51 new issues in total, of which 8 have been verified and 11 have been addressed by developers. These results support our hypothesis that the suggested method might be helpful in finding new Android app issues.

Table 2: Related work summarizing the key aspects of each literature review on software testing advancements.

Reference	Aim	Achievements	Drawbacks	Future Work
[26]	Review of Agile software testing methodologies and their integration in Agile development.	Demonstrated that Agile testing is an essential, integrated part of development, using Agile Testing Quadrants for test organization.	Limited focus on non-functional testing in Agile environments.	Incorporation of enhanced testing procedures for improved Agile development.

International Journal of Core Engineering & Management
Volume-7, Issue-09, 2024 ISSN No: 2348-9510

[27]	Explore the role of Artificial Intelligence (AI) in improving software testing processes.	AI-driven testing demonstrated enhanced quality assurance and reduced time, increasing organizational efficiency in software development.	Lacks in-depth analysis of AI's limitations in testing complex systems.	Application of smarter AI-based testing methods for more sophisticated software applications.
[28]	Development of a simulation-based framework for performance, scalability, and security testing.	Significant improvements in testing efficiency (18.75%-time reduction), CPU and memory utilization, and scalability testing through realistic environment simulations.	Simulation complexity can lead to challenges in replicating specific real-world conditions accurately.	Further optimization of the framework for diverse cloud environments and more advanced security testing.
[29]	Introduction of Evo Crash, an automated crash reproduction tool using a Guided Genetic Algorithm (GGA).	Outperformed state-of-the-art crash reproduction techniques and assisted developers in quicker debugging and more frequent fixes.	Requires a significant amount of computational power and time to generate the necessary crash reproduction tests.	Improving the tool's efficiency and exploring further integration with continuous integration (CI) pipelines.
[30]	Use of machine learning to identify potential software defects, considering component dependency.	Identified defect-prone areas more accurately by utilizing a component dependency score, leading to improved control over software quality in evolving systems.	The model may struggle with rapidly changing software dependencies.	Enhancing the machine learning model to adapt to dynamically evolving software more efficiently.
[31]	Analysis of challenges in Machine Learning (ML)-enabled software development.	Addressed mismatch-detection through stakeholder alignment, integrated end-to-end pipelines, and reframed traditional Software Engineering (SE) practices for ML.	Collaboration between SE and ML teams requires improvement for greater efficiency.	Strengthen education, training, and cooperation between stakeholders, and further automate mismatch detection in complex ML-enabled systems.
[32]	Collaborative bug-finding approach in Android app testing using Bugine for GitHub issue recommendation.	Introduced Bugine, which helped novice testers discover 34 new bugs using NLP and ranking algorithms, significantly improving the bug discovery process in Android apps.	Time-consuming search process for bug reports before implementing Bugine.	Further enhancements to Bugine's algorithms to improve accuracy in identifying and fixing high-priority bugs for app development.

International Journal of Core Engineering & Management
Volume-7, Issue-09, 2024 ISSN No: 2348-9510

VII. CONCLUSION AND FUTURE WORK

Modern software development would not be possible without test automation frameworks, which have completely transformed the testing process. Selenium, TestNG, Cucumber, JMeter, and Cypress are just a few of the frameworks that provide a structured approach to testing, which in turn makes it faster, more accurate, and more efficient. Test frameworks automate repetitive operations, which minimize testing time and free up engineers to focus on increasing software quality and functionality. Together, its useful features—such as thorough reporting, reusable test scripts, and real-time feedback—improve program maintainability and cut down on manual effort. And to find and fix defects early in the development cycle, automation frameworks are essential to continuous delivery (CD) and continuous integration (CI) pipelines. Faster release cycles, lower maintenance costs, and more dependable software are the results of this. Functional and performance testing are essential, but frameworks such as JMeter are even more so for finding performance bottlenecks and making software even more stable. An intriguing new frontier is the merging of testing frameworks. Organizations can ensure high-quality software products and flawless user experiences by utilizing these technologies to improve accuracy, test coverage, and tackle complicated software development challenges.

The future of test automation is now in combination with AI testing techniques and Large Language Models for automatic test-case development. Continuing complexity of software systems makes the need for more intelligent and autonomous automation frameworks grow. Studies in fine tuning LLMs for certain tasks such as mobile application testing along with the development and improvements in cloud-based testing will provide more efficient and scalable testing solutions. The investigation into distributed testing environments and the simulation of particular systems will improve the accuracy of test automation and accommodate more considerable software systems at a time. Furthermore, the enhancement of ML for software testing will also overcome some specific issues in refining the process and result in a more intelligent and self-sufficient testing process.

REFERENCES

1. H. V. Gamido and M. V. Gamido, “Comparative review of the features of automated software testing tools,” Int. J. Electr. Comput. Eng., 2019, doi: 10.11591/ijece.v9i5.pp4473-4478.
2. M. S. M. Ms Vinita Rohilla, “Software Verification and Validation,” Softw. Verif. Valid., 2008.
3. A. Mishra and Y. I. Alzoubi, “Structured software development versus agile software development: a comparative analysis,” Int. J. Syst. Assur. Eng. Manag., 2023, doi: 10.1007/s13198-023-01958-5.
4. Hannonen A, “AUTOMATED TESTING FOR SOFTWARE PROCESS AUTOMATION,” Univ. VAASA, 2020.
5. R. P. Vamsi Krishna Yarlagadda, “Secure Programming with SAS: Mitigating Risks and Protecting Data Integrity,” Eng. Int., vol. 6, no. 2, pp. 211–222, 2018.
6. D. P. Wangoo, “An Intelligent Journey to Machine Learning Applications in Component-Based Software Engineering,” 2020. doi: 10.1007/978-981-15-0222-4_16.
7. K. Beck, Extreme Programming Explained: Embrace Change. 1999.
8. V. K. Y. Nicholas Richardson, Rajani Pydipalli, Sai Sirisha Maddula, Sunil Kumar Reddy Anumandla, “Role-Based Access Control in SAS Programming: Enhancing Security and Authorization,” Int. J. Reciprocal Symmetry Theor. Phys., vol. 6, no. 1, pp. 31–42, 2019.
9. R. S. Pressman, “Software engineering: A practitioner’s approach,” Adv. Eng. Softw., vol. 5, no. 3, p. 171, Jul. 1982, doi: 10.1016/0141-1195(83)90118-3.

10. M. Bohme and S. Paul, "A Probabilistic Analysis of the Efficiency of Automated Software Testing," IEEE Trans. Softw. Eng., 2016, doi: 10.1109/TSE.2015.2487274.
11. M. Edgar Serna, M. Raquel Martínez, and O. Paula Tamayo, "A review of reality of software test automation," Comput. y Sist., 2019, doi: 10.13053/CyS-23-1-2782.
12. R. Dwivedi and V. Rohilla, "Empowering agile method feature-driven development by extending it in RUP shell," in Advances in Intelligent Systems and Computing, 2017. doi: 10.1007/978-981-10-3770-2_69.
13. . C. Jorgensen, Software Testing: A Craftsman's Approach: Fourth Edition. 2013. doi: 10.1201/b15980.
14. B. Oliinyk and V. Oleksiuk, "Automation in software testing, can we automate anything we want?," in CEUR Workshop Proceedings, 2019.
15. S. Berner, R. Weber, and R. K. Keller, "Observations and lessons learned from automated testing," in Proceedings - 27th International Conference on Software Engineering, ICSE05, 2005. doi: 10.1145/1062455.1062556.
16. W. D. van Driel, J. W. Bikker, M. Tijink, and A. Di Bucchianico, "Software reliability for agile testing," Mathematics, 2020, doi: 10.3390/MATH8050791.
17. A. M. Kale, V. V Bandal, and K. Chaudhari, "a Review Paper on Software Testing Techniques and Tools," Int. Res. J. Eng. Technol., 2019.
18. S. Izzat and N. N. Saleem, "Software Testing Techniques and Tools: A Review," J. Educ. Sci., 2023, doi: 10.33899/edusj.2023.137480.1305.
19. Vaishnavi S Kulkarni and Asha N, "Developing a Robust Framework for Test Automation," Int. J. Eng. Res., 2015, doi: 10.17577/ijertv4is060677.
20. "Automation of Performance Testing: A Review," Int. J. Intell. Comput. Inf. Sci., 2022, doi: 10.21608/ijicis.2022.161846.1219.
21. C. Rankin, "The software testing automation framework," IBM Syst. J., 2002, doi: 10.1147/sj.411.0126.
22. K. Eldrandaly, M. A. ElLatif, and N. Zaki, "Comparative Study of Software Test Automation Frameworks," Int. J. Eng. Trends Technol., 2019, doi: 10.14445/22315381/IJETT-V67I11P216.
23. Rohit Khankhoje, "An In-Depth Review of Test Automation Frameworks: Types and Trade-offs," Int. J. Adv. Res. Sci. Commun. Technol., 2023, doi: 10.48175/ijarsct-13108.
24. S. AMARICAI and R. CONSTANTINESCU, "Designing a Software Test Automation Framework," Inform. Econ., 2014, doi: 10.12948/issn14531305/18.1.2014.14.
25. K. Khaerunnisa, N. Selviandro, and R. R. Riskiana, "Comparative Study of Robot Framework and Cucumber as BDD Automated Testing Tools," Ultim. J. Tek. Inform., 2023, doi: 10.31937/ti.v15i1.3228.
26. K. Pathak, S. Ninoria, and S. Bharadwaj, "Scope of Agile Approach for Software Testing Process," in Proceedings of the 2022 11th International Conference on System Modeling and Advancement in Research Trends, SMART 2022, 2022. doi: 10.1109/SMART55829.2022.10047649.
27. P. Singhal, S. Kundu, H. Gupta, and H. Jain, "Application of Artificial Intelligence in Software Testing," in 2021 10th International Conference on System Modeling & Advancement in Research Trends (SMART), 2021, pp. 489–492. doi: 10.1109/SMART52563.2021.9676244.
28. P. Thantharate, "SCALE-IT: Distributed and Realistic Simulation Frameworks for Testing Cloud-Based Software," in International Conference on Electrical Engineering, Computer Science and Informatics (EECSI), 2023. doi: 10.1109/EECSI59885.2023.10295630.
29. M. Soltani, A. Panichella, and A. Van Deursen, "Search-Based Crash Reproduction and Its Impact on Debugging," IEEE Trans. Softw. Eng., 2020, doi: 10.1109/TSE.2018.2877664.

30. S. Sutar, R. Kumar, S. Pai, and B. R. Shwetha, "Defect Prediction based on Machine Learning using System Test Parameters," in Proceedings - 2019 Amity International Conference on Artificial Intelligence, AICAI 2019, 2019. doi: 10.1109/AICAI.2019.8701345.
31. S. Saeed, M. M. Abubakar, and M. Karabatak, "Software Engineering for Data Mining (ML-Enabled) Software Applications," in 9th International Symposium on Digital Forensics and Security, ISDFS 2021, 2021. doi: 10.1109/ISDFS52919.2021.9486319.
32. . H. Tan and Z. Li, "Collaborative bug finding for android apps," in Proceedings - International Conference on Software Engineering, 2020. doi: 10.1145/3377811.3380349.