Available online www.ejaet.com

European Journal of Advances in Engineering and Technology, 2020, 7(9):78-84

Research Article ISSN: 2394 - 658X

Automating Software Testing: A Comparative Study of Machine Learning and Traditional Approaches

Kodanda Rami Reddy Manukonda

reddy.mkr@gmail.com

ABSTRACT

This abstract provides a thorough analysis of the differences between standard software testing process automation methods and machine learning (ML) techniques. The need for effective testing approaches has increased due to the software development processes' quick evolution. Conventional testing techniques, however dependable, frequently find it difficult to keep up with the complexity and size of contemporary software systems. On the other hand, machine learning approaches present viable ways to automate certain parts of software testing by using algorithms to enhance the creation of test cases, identify abnormalities, and analyze patterns. We compare the efficacy, efficiency, and scalability of machine learning (ML)-based testing to traditional methods across a variety of software development scenarios in this paper. We assess test coverage, fault detection rates, resource utilization, and maintenance overhead through a series of experiments and case studies. Our research offers insightful information about the relative advantages and drawbacks of ML versus conventional techniques, illuminating the best practices for incorporating machine learning into software testing processes. By helping practitioners and researchers make well-informed decisions when choosing the best techniques for their unique testing requirements, this research advances the state-of-the-art in automated testing methods.

Keywords: software testing, testing automation, machine learning, artificial intelligence, and software testing tools

---

INTRODUCTION

Ensuring the reliability and quality of software frameworks is crucial in the dynamic field of software development. Software testing is a crucial component of this cycle that aims to identify defects, errors, and vulnerabilities in software programs [1]. Software testing has always been a laborious and demanding process that frequently calls for physical labor and extensive resources. That being said, there is growing interest in automating software testing cycles to increase efficiency and viability as a result of advances in machine learning (ML) [2]. This study offers a comparative analysis of machine learning and traditional approaches to automating software testing, highlighting their respective benefits, drawbacks, and potential uses [3].

A fundamental cycle of software development, software testing aims to identify errors and ensure that the program operates as intended [4]. It entails running the software in a controlled manner to verify that it satisfies the requirements and to find any discrepancies between expected and actual behavior [5]. Software testing has traditionally been thought of as a specific step in framework improvement, but it is increasingly seen to be a continuous activity that needs to be managed throughout the plan, improvement, and support stages [6].

Integration of Manual and Automated Testing Strategies

Using the advantages of each approach, this research suggests a tool to coordinate testing methods that are both mechanical and manual. The general testing cycle can benefit from both the expertise and reproducibility of automated testing as well as the human knowledge and instinct of manual testing by combining the two types of

testing. Additionally, integrating computer testing into the process takes into account the basis of relapse testing data sets, utilizing the repurposing of ensuing tests to identify bugs. Additionally, for both human and automated tests, proportions of inclusion, such as code, dataflow, and detail inclusion, can be reported, providing a comprehensive assessment of the testing system [7].

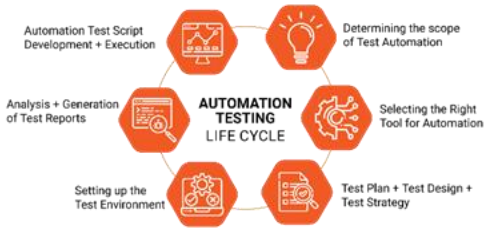

Figure 1: Automation Testing Cycle

- **Manual Testing**
  Human analysts carry out tests as part of manual testing, focusing on examining the functionality and behaviour of the software. Common manual testing systems include unit, module, sub-framework, and framework testing. While module testing focuses on evaluating assortments of ward pieces, unit testing verifies specific program components. Sub-framework testing involves evaluating various combinations of modules, whereas framework testing certifies that the framework as a whole satisfies its non-utilitarian and practical requirements. Acknowledgment testing is part of alpha testing, the final phase prior to functional use, which ensures that the framework handles client issues [8].
- **Automated Testing**
  On the other hand, automated testing relies on scripts and software tools to carry out tests. Mechanized testing provides competency and repeatability, whereas manual testing brings the benefits of human judgment and instinct. Robotized testing accelerates the testing system and enables faster input cycles by computerizing boring and laborious testing assignments. However, there may be limitations to computerized testing in terms of test inclusion and its ability to identify specific types of flaws [9].

Understanding Traditional Software Testing Approaches

The planning, carrying out, and evaluating of manual experiments are standard procedures used in software testing. This cycle entails designing tests based on predetermined requirements and conclusions, followed by carrying out these experiments to find flaws or shortcomings in the program that is being tested. Manual testing methodologies, such as white-box, black-box, and joining testing, rely on human intuition and skill to identify potential problems in software systems. Even though traditional testing methods have been widely used and shown to be effective in most circumstances, they are frequently constrained by factors like inclusion, repeatability, and adaptability [10].

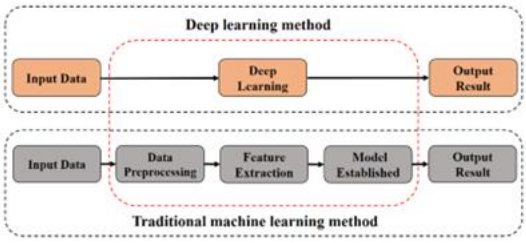

Figure 2: Comparison Of Deep Learning and Traditional Machine Learning Methods

The Role of Machine Learning in Automating Software Testing

Technological advances in machine learning present intriguing solutions to computerize several aspects of software testing, addressing many of the challenges associated with traditional approaches. To organically generate trials, concentrate on test execution, and identify instances or anomalies typical of potential software flaws, ML computations can be prepared on real test data. Through the use of machine learning techniques such

as controlled learning, unassisted learning, and support learning, computerized testing instruments can improve with time in terms of accuracy and suitability. Additionally, ML-based testing techniques may be able to scale across large, intricate software frameworks, enabling more thorough test inclusion and faster input cycles.

Objectives of the study

1. To evaluate the effectiveness and efficiency of machine learning-based automated testing compared to traditional manual testing approaches
2. To identify the strengths and limitations of both machine learning and traditional approaches in automating software testing.

LITERATURE REVIEW

Arcuri et.al (2019) address the fundamental problem of assuring the integrity and robustness of RESTful APIs in modern software development. This is a problem that has become increasingly important in recent years. When it comes to allowing communication between different dispersed system parts, RESTful application programming interfaces (APIs) are vital components. However, evaluating their functionality in a thorough manner presents a considerable difficulty. In the field of software testing procedures, particularly in the realm of web services and distributed systems, the launch of EvoMaster by Arcuri represents a revolutionary advancement. EvoMaster is able to generate a wide variety of test cases that exercise a variety of functionalities and edge cases by utilizing evolutionary algorithms. This allows EvoMaster to methodically investigate the input space of RESTful APIs. Because of this rigorous approach, not only is test coverage improved, but also effectiveness is increased, which ultimately results in an improvement in the overall quality and dependability of RESTful API implementations. The study conducted by Arcuri is considered to be a cornerstone in the area. It provides researchers and practitioners alike with a strong tool that may guarantee the correctness and resilience of RESTful API-based systems or applications [11].

Marginean et.al (2019) makes a substantial contribution to the field of software maintenance and repair by presenting a unique method to the problem under investigation. When it comes to large software systems, the presence of software defects is a significant risk to both the stability and performance of the system. The conventional approaches of debugging and repair can prove to be arduous and time-consuming, particularly in the context of large-scale software projects. By introducing Sapfix, an automated repair tool that is meant to find and resolve issues in an end-to-end way, Marginean and the team are able to tackle this difficulty head-on. Sapfix is able to autonomously diagnose, localize, and resolve software issues with minimal assistance from humans. This is accomplished by utilizing machine learning techniques and drawing from huge repositories of code. The process of repairing software is simplified as a result of this paradigm change in software maintenance, which also helps to reduce downtime and improve overall system reliability. The work that was done by Marginean and colleagues represents a key milestone in automated software repair approaches. It provides a solution that is scalable and was designed to reduce the impact that software faults have on complex systems [12].

Calò et.al (2020) research, which is very important. The importance of guaranteeing the safety and dependability of these systems cannot be overstated in light of the growing number of autonomous vehicles on the road. In the process of testing autonomous driving systems, one of the most significant issues is the generation of test scenarios that are both realistic and thorough, covering a wide variety of possible vehicle driving scenarios. In order to tackle this difficulty, Calò and colleagues have proposed a system that may generate collision scenarios that can be avoided, which would allow for comprehensive testing of algorithms used in autonomous vehicles. Their technique makes it easier to identify probable failure sources and validate collision avoidance capabilities in autonomous cars by modeling real-world scenarios in controlled conditions. This is accomplished through the use of simulation. This research makes a significant contribution to the field of autonomous systems testing by providing novel insights and approaches. These findings have the potential to improve the safety and dependability of vehicles that drive themselves [13].

Abdessalem et.al (2018) investigates the testing of autonomous vehicles to determine whether or not there are feature interaction shortcomings. Additionally, as the technology behind autonomous vehicles continues to progress, the complexity of these systems continues to increase, which in turn increases the potential of unanticipated interactions between various elements. It is possible for feature interaction failures to have

significant repercussions for the safety and dependability of autonomous driving systems. For the purpose of systematically identifying and mitigating feature interaction errors in autonomous vehicles, Abdessalem et al. suggest a search strategy that utilizes multiple objectives. Their method, which makes use of evolutionary algorithms and multi-objective optimization techniques, makes it possible to detect and resolve feature interaction difficulties in an effective manner, hence contributing to the overall dependability of autonomous vehicle systems. Through this research, a significant part of testing autonomous systems is addressed, and practical solutions are provided for ensuring the safe and reliable operation of these systems in environments that are representative of the actual world [14].

Goues et.al (2019) present a comprehensive and incisive overview of automated program repair approaches. These techniques are essential in minimizing the widespread problem of software flaws in the field of software development. Due to the fact that software flaws can result in system crashes, vulnerabilities, and a variety of other negative effects, the requirement for efficient bug-fixing techniques is of the utmost importance. Automated program repair is a potential option since it enables the automatic identification and correction of software errors without the need for human interaction during the process. A wide variety of automated repair approaches, such as program synthesis, fault localization, and automated testing, are methodically investigated by Goues and the team. They investigate the advantages, disadvantages, and prospective applications of each of these techniques. The process of automatically producing computer code to correct deficiencies that have been identified is referred to as program synthesis. Fault localization techniques are utilized to assist in identifying the underlying causes of software faults. The validation and verification of the efficiency of the repair operations are both significantly aided by the utilization of automated testing approaches. Providing insights and methodologies to improve software quality, reliability, and robustness through automated bug-fixing mechanisms, the research conducted serves as a valuable resource for both software developers and researchers. This is because the research elucidates these techniques and discusses the practical implications of these techniques. This thorough review not only makes a contribution to the theoretical knowledge of automated program repair, but it also offers practical assistance for utilizing these techniques in real-world software development scenarios, thereby contributing to the advancement of the state-of-the-art in software engineering practices [15].

COMPARATIVE STUDY OF MANUAL VS AUTOMATED TESTING

Automated vs. Manual Testing

There are two main approaches that are used in software quality assurance: automated testing and manual testing. These techniques show different ways to find flaws and guarantee the performance and dependability of software.

Comparative Analysis of Manual Testing

- **Time-Consuming Nature:** Because manual testing depends on human interaction at every stage of the testing process, it frequently requires a significant time and effort commitment.
- **Lack of Learning Opportunities:** Because manual test execution may not present testers with cutting-edge obstacles or technological advancements, manual testing may not provide ongoing learning opportunities.
- **Risk of Neglect:** There is a real possibility that manual testing will be neglected, which could result in insufficient test coverage and possibly miss important software flaws.
- **Difficulty in Test Maintenance:** Keeping an extensive list of manual tests up to date and making sure they are consistently run may be very difficult, especially as software programs change.
- **Limited Reusability:** Because manual tests must be repeated for several stakeholders, they are usually not reusable across testing cycles, leading to redundant work.
- **Integration Testing Bias:** Inadvertently, manual testing may prioritize integration testing above the isolation and testing of separate software modules or components.
- **Lack of Scripting Facilities:** Because scripting is not integrated into manual testing, it is difficult to automate repetitious test jobs and situations.

Advantages of Automated Testing

- **Speed:** When compared to human users, automated tests have substantially faster execution times, which allows for timely feedback on software quality.
- **Reliability:** Automated tests reduce the possibility of human error and provide reliable test results by constantly carrying out the same activities with each execution.
- **Programmability:** Sophisticated testing scenarios can be written to be executed by automated tests, which can reveal latent problems and offer important insights into the behavior of software.
- **Comprehensiveness:** Automation makes it possible to create exhaustive test suites that cover every aspect of a program, guaranteeing complete test coverage.
- **Reusability:** Because automated tests may be reused in several testing cycles, they reduce the time and effort needed for repeated testing jobs.

Cost Model Based Analysis

Opportunity Cost Consideration: To balance automated and manual testing under budgetary constraints, a linear optimization-based cost model is suggested. The opportunity cost of automating test cases as opposed to carrying out manual test cases is taken into consideration by this methodology.

Real Example and Scenario Analysis

A specific model is provided to illustrate the potential applications of the suggested cost model under various testing scenarios. This model takes into account things like the benefits of automated testing based on chance openness and the goal of increasing test inclusion through manual testing.

Scenario A: In this case, the objectives are to test at least one delivery in its entirety and to survey the most fundamental 50% of the framework across all deliveries. These objectives align with constraints R3.1 and R2.2. Figure 3 illustrates the optimal configuration, which is located at point S1 (na = 50, nm = 100) on the creation prospects wilderness denoted by R1. Thus, all trials should undergo manual testing once, and 50 experiments related to the most fundamental 50% of the framework should be computerized.

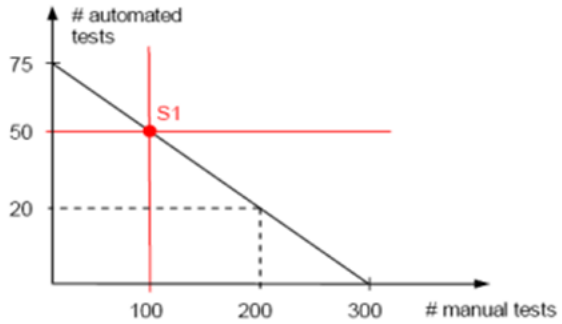

Figure 3: Scenario of Auto vs. Manual A

Scenario B: The goals are to test about one delivery in its whole and evaluate the lowest 20% of the framework for every delivery. These objectives are related to constraints R3.1 and R2.1. Any point inside Figure 4's hidden area satisfies these requirements. The objective capability, nevertheless, ensures that the optimal configuration is between focuses S1 (na = 50, nm = 100) and S2 (na = 20, nm = 220) on the creation prospective wilderness represented by R1.

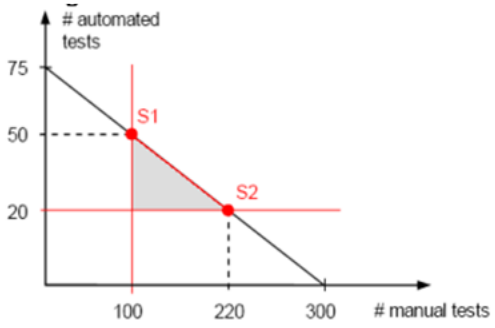

Figure 4: Scenario of Auto vs. Manual B.

Scenario C: This scenario aims to assess the most fundamental 50% of the framework for all deliveries and test around two deliveries in total. Goals R3.2 and R2.2 are managed by limits. However, as shown in Figure 5, there isn't a solution that satisfies the two constraints.

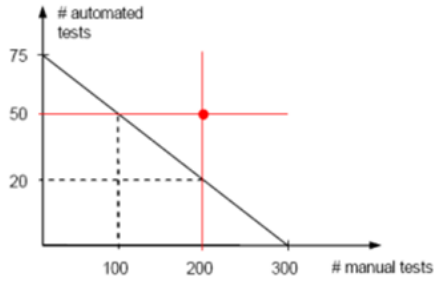

Figure 5: Scenario of Auto vs. Manual C.

FUTURE SCOPE

The field of software testing is constantly changing, and the use of machine learning (ML) techniques in conjunction with more conventional methods could lead to exciting developments in the future. This comparative analysis predicts that machine learning (ML) techniques will improve testing efficiency through task automation, faster anomaly and vulnerability detection, and support for adaptable testing approaches. ML can maximize test coverage, forecast possible problems, and improve the overall quality of software products by utilizing large datasets. In addition, machine learning's capacity to draw lessons from previous testing encounters might result in proactive risk reduction and ongoing enhancement, making it a crucial instrument in the development of dependable and durable software systems.

CONCLUSION

In software testing, there is always a constant pursuit of efficiency, accuracy, and thorough coverage. "Automating Software Testing: A Comparative Study of Machine Learning and Traditional Approaches" provides insightful information about the subtle factors that go into developing test strategies. By means of a thorough analysis, the research elucidates the various benefits and drawbacks that are intrinsic to both automated and manual testing approaches. With its superior speed, dependability, and programmability made possible by specialized tools and scripting capabilities, automated testing proves to be a powerful ally. On the other hand, manual testing is still necessary in situations when human intuition and sophisticated analysis are needed, even though it is labor-intensive. The research highlights the significance of striking a balance between automated and manual testing while adhering to financial restrictions. It also highlights the use of advanced cost models and scenario analysis to maximize resource use. Moreover, it highlights the revolutionary potential of cutting-edge technologies such as machine learning, which have the ability to completely change traditional testing paradigms through the improvement of predictive analysis and test automation. In the end, the study promotes a context-driven methodology for software testing and the deployment of cutting-edge technologies in tandem with the automated and manual approaches' synergistic integration. With this strategy, businesses can confidently manage the complexity of contemporary software development and produce software solutions that satisfy end users' changing needs while maintaining high quality and resilience.

REFERENCES

[1]. Durelli, V. H., Durelli, R. S., Borges, S. S., Endo, A. T., Eler, M. M., Dias, D. R., & Guimarães, M. P. (2019). Machine learning applied to software testing: A systematic mapping study. *IEEE Transactions on Reliability*, *68*(3), 1189-1212.

[2]. Braiek, H. B., & Khomh, F. (2020). On testing machine learning programs. *Journal of Systems and Software*, *164*, 110542.

[3]. Riccio, V., Jahangirova, G., Stocco, A., Humbatova, N., Weiss, M., & Tonella, P. (2020). Testing machine learning based systems: a systematic mapping. *Empirical Software Engineering*, *25*, 5193-5254.

[4]. Harer, J. A., Kim, L. Y., Russell, R. L., Ozdemir, O., Kosta, L. R., Rangamani, A., ... & Lazovich, T. (2018). Automated software vulnerability detection with machine learning. *arXiv preprint arXiv:1803.04497*.

[5]. Jan, B., Farman, H., Khan, M., Imran, M., Islam, I. U., Ahmad, A., ... & Jeon, G. (2019). Deep learning in big data analytics: a comparative study. *Computers & Electrical Engineering*, *75*, 275-287.

[6]. Kong, P., Li, L., Gao, J., Liu, K., Bissyandé, T. F., & Klein, J. (2018). Automated testing of android apps: A systematic literature review. *IEEE Transactions on Reliability*, *68*(1), 45-66.

[7]. Takanen, A., Demott, J. D., Miller, C., & Kettunen, A. (2018). *Fuzzing for software security testing and quality assurance*. Artech House.

[8]. Le, V. M., Felfernig, A., Uta, M., Benavides, D., Galindo, J., & Tran, T. N. T. (2021, May). DirectDebug: Automated testing and debugging of feature models. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)* (pp. 81-85). IEEE.

[9]. Tian, Y., Pei, K., Jana, S., & Ray, B. (2018, May). Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th international conference on software engineering* (pp. 303-314).

[10]. Liu, K., Koyuncu, A., Kim, D., & Bissyandé, T. F. (2019, July). Tbar: Revisiting template-based automated program repair. In *Proceedings of the 28th ACM SIGSOFT international symposium on software testing and analysis* (pp. 31-42).

[11]. Arcuri, A. (2019). RESTful API automated test case generation with EvoMaster. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *28*(1), 1-37.

[12]. Marginean, A., Bader, J., Chandra, S., Harman, M., Jia, Y., Mao, K., ... & Scott, A. (2019, May). Sapfix: Automated end-to-end repair at scale. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)* (pp. 269-278). IEEE.

[13]. Calò, A., Arcaini, P., Ali, S., Hauer, F., & Ishikawa, F. (2020, October). Generating avoidable collision scenarios for testing autonomous driving systems. In *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)* (pp. 375-386). IEEE.

[14]. Abdessalem, R. B., Panichella, A., Nejati, S., Briand, L. C., & Stifter, T. (2018, September). Testing autonomous cars for feature interaction failures using many-objective search. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering* (pp. 143-154).

[15]. Goues, C. L., Pradel, M., & Roychoudhury, A. (2019). Automated program repair. *Communications of the ACM*, *62*(12), 56-65.