ISSN No:-2456-2165

Machine Learning for Automation Software Testing Challenges, Use Cases Advantages & Disadvantages

Ashritha S
Dept. of Information & Science Engineering
R v College of Engineering
Bangalore, India

Dr. Padmashree T
Dept. of Information & Science Engineering
R v College of Engineering
Bangalore, India

Abstract:- Software testing is a method for checking and validating an automated system's ability to fulfill the automation's necessary attributes and functionality with the automation. It is an essential part of software development that is vital to ensuring the quality of a product to be released. The need for automated software testing approaches arises as the operating structures become more complex which requires analyzing software systems behavior to discover faults. Many testing activities are expensive and complex, and the automation of software testing is a realistic approach that has been implemented to get around these problems. At the beginning, when the Waterfall project approach was already commonly applied, testing was introduced to validate the program as an end-of-project solution only before it entered the market. Since then, project methodologies have also evolved, integrating the ever-popular Agile, DevOps, and others, requiring more versatile and innovative methods. Machine learning (ML) is one of the new approach introduced to use the groundbreaking technology made possible. Machine Learning is established from the study of pattern recognition and computational learning approach. The main principle reason is to make machines learn without being explicitly programmed. This science absorbs tons of complex data and identifies schemes that are predictive. In this paper, review the state-of-the-art ways in which ML is explored for automating and upgrading software testing is set. And include an overview of the use cases of test automation, an advantage in implementing ML automation techniques along with challenges in current automation testing. The aftereffects of this paper plot the ML viewpoint that are most regularly used to automate software-testing exercises, helping analysts to comprehend the ebb and flow condition of research concerning ML applied to software testing. Its strategies have demonstrated to be very valuable for this automation process and there has been a developing enthusiasm for applying machine learning to mechanize different software testing activities.

Keywords:- *Machine learning (ML), Software testing, Challenges, Use cases of ML based test automation.*

I. INTRODUCTION

Most early software applications belonged to the scientific computing and data processing domains [1]. Over the past few decades, however, there has been a substantial growth in the software industry, which was primarily driven by advances in technology. Consequently, software has become increasingly important in modern society. As software becomes more pervasive in everyday life, software engineers must meet stringent requirements to obtain reliable software. To keep up with all these advances, software engineering has come a long way since its inception. Yet, many software projects still fail to meet expectations due to a combination of factors as, for instance, cost overruns and poor quality. Evidence suggests that one of the factors that contribute the most to budget overruns is fault detection and fault correction: uncorrected faults become increasingly more expensive as software projects evolve. To mitigating such overheads, there has been a growing interest in software testing, which is the primary method to evaluate software under development [3].

Software testing plays a pivotal role in both achieving and evaluating the quality of software. Despite all the advances in software development methodologies and programming languages, software testing remains necessary. Basically, testing is a process whose purpose is to make sure that the software artifacts under test do what they were designed to do and that they do not do anything unintended, thus raising the quality of these artifacts [4]. Nevertheless, testing is costly, resource consuming, and notoriously complex: studies indicate that testing accounts for more than 50% of the total costs of software development [5]. Moreover, as any human-driven activity, testing is error-prone and creating reliable software systems is still an open problem. In hopes of coping with this problem, researchers and practitioners have been investigating more effective ways of testing software. A practical strategy for facing some of the issues is to automate software testing. Thus, a lot of effort has been put into automating testing activities. ML techniques have been successfully used to reduce the effort of carrying out many software engineering activities [8]. Machine learning [9], which is a research field at the intersection of ML, computer science, and statistics, has been applied to automate various software engineering activities [10]. It turns out that some software-testing issues lend themselves to being formulated as learning problems and tackled by learning algorithms, so there has been a growing interest in capitalizing on ML to automate and streamline software testing. In addition, software

systems have become increasingly complex, so some conventional testing techniques may not scale well to the complexity of these modern software systems. This ever-increasing complexity of modern software systems has rendered ML-based techniques attractive.

II. BACKGROUND

This section covers background on software testing and ML. The discussion is divided into two parts: the first covers the pure- pose of software testing, giving special emphasis to elucidating the most fundamental concepts; the second part lays out the essential background on ML.

A. Software Testing

Software testing is a quality confirmation action that comprises in assessing the framework under test by watching its execution with the point of uncovering failures. A disappointment is identified when the outer conduct is not quite the same as what is anticipated from the test as indicated by its prerequisites or some other depiction of the normal conduct [3]. Since this action re-quires the execution, it is regularly alluded to as powerful examination. Conversely, there are quality affirmation exercises that don't require the execution [5].

A significant component of the testing action is the experiment. Basically, an experiment indicates in which conditions of the test must be executed to find a failure. At the point when an experiment uncovers a disappointment, it is viewed as fruitful (or successful). An experiment typifies the information esteems expected to execute the test [3]. Along these lines, experiment inputs change in nature, going from client contributions to technique calls with the experiment esteems as parameters. To assess the aftereffects of experiments, analyzers must realize what yield the test would deliver for those experiments. The component that checks the accuracy of the yields delivered is alluded to as prophet. For the most part, analyzers assume the job of prophet. Be that as it may, it merits stressing that a prophet can be a determination or much another program.

Executing experiments physically is expensive, tedious, and mistake inclined. In this manner, many testing systems and instruments have been created along the years with the goal of supporting the mechanized execution of experiments at various levels. Testing systems that help unit testing have been generally utilized particularly because of the advancement of nimble philosophies and test-centered methodologies. More as of late, many record-and-play or even content based systems and instruments to perform graphical UI (GUI) testing have gotten increasingly well known among engineers. Even though automating the execution of experiments spoke to a huge improvement in the field, programming testing exercises will in general become progressively troublesome and exorbitant as frameworks become progressively increasingly mind boggling. The exemplary answer of programming specialists to lessen cost and unpredictability is automation. Consequently, in the previous scarcely any years, numerous endeavors have been done to think of robotized approaches for creating test information sources and upgrades to meet distinctive test objectives (e.g., branch inclusion). Three distinct methods to produce experiments consequently hang out in this situation: representative execution, search based, and irregular methodologies [15].

B. Machine Learning

Essentially, problem solving using computers revolves around coming up with algorithms, which are sequences of instructions that when carried out turn the input (or set of inputs) into an output (or set of outputs). For instance, many algorithms for sorting have been proposed over the years. As input, these algorithms take a set of elements (e.g., numbers) and the output is an ordered list (e.g., list of numbers in ascending or descending order).

Many problems, however, do not lend themselves well to being solved by traditional algorithms. An example of problem that is hard to solve through traditional algorithms is predicting whether a test case is effective. Depending on the test, the input is like: for instance, for a program that implements a sorting algorithm, it is a list of elements (e.g., numbers). Also know what the output should be: an ordered list of elements. Nevertheless, the test does not know what list of elements is most likely to uncover faults: that is, what inputs will exercise different parts of the program's code.

There are many problems for which there is no algorithm. In effect, trying to solve these problems through traditional algorithms has led to limited success. However, in recent years, a vast amount of data concerning such problems has become available. This rise in data availability has prompted researchers and practitioners to look at solutions that involve learning from data: ML algorithms.

Apart from the explosion of data being captured and stored, the recent widespread adoption of ML algorithms has been largely fueled by two contributing factors: first, the exponential growth of compute power, which has made it possible for computers to tackle ever-more-complex problems using ML, and second, the increasing availability of powerful ML tools [16], [17]. Due to these advances, researchers and practitioners have applied ML algorithms to an ever-expanding range of domains. Web search engines, natural language processing, speech recognition, computer vision, and robotics [18]- [20]. It is worth noting, however, that ML is not new. As pointed out by Louridas and Ebert [21], ML has been around since the 1970s, when the first ML algorithms emerged.

Let us go back to the problem of predicting the effectiveness of test cases. When facing problems of this nature, data come into play when it is needing to know what an effective test case looks like. Although we might not know how to come up with an effective test case, we assume that some effective test cases will be present in the collected data (e.g., set of inputs for a program whose run-time behavior was also recorded). If an ML algorithm can learn from the available test-case data, and if the program under test did not deviate much from the version used during data collection, it is possible to make predictions based on the results of the

algorithm. Although the ML algorithm may not be able to identify the whole test-case evaluation process, it can still detect some hidden structures and patterns in the data. In this context, the result of the algorithm is an approximation (i.e., a model). In a broad sense, ML algorithms process the available data to build models. The resulting models embody patterns that allow us to make inferences and better characterize problems as predicting the effectiveness of test cases.

At its core, ML is simply a set of algorithms for designing models and understanding data [19], [20]. Therefore, as stated by Mohri et al. [18], ML algorithms are data-driven methods that combine computer science concepts with ideas from statistics, probability, and optimization. As emphasized by Shalev- Shwartz and Ben-David [22], the main difference in comparison with traditional statistics and other fields is that in computer science, ML is centered around learning by computers, so algorithmic considerations are key.

Many ML algorithms have been devised over the years. Essentially, these ML algorithms differ in terms of the models they use or yield. These algorithms can be broadly classified as supervised or unsupervised.

Software has been playing an increasingly important role in modern society. Therefore, ensuring software quality is vital. Although many factors impact the development of reliable soft- ware, testing is the primary approach for assessing and improving software quality [3]. However, despite decades of research, testing remains challenging. Recently, a strategy that has been adopted to circumvent some of the open issues is applying ML algorithms to automate software testing. We set out to provide an overview of the literature on how researchers have harnessed ML algorithms to automate software testing. We detail the rationale behind our research in the following sections.

III. PROBLEM STATEMENT AND JUSTIFICATION

Although applying ML to tackle software-testing problems is a relatively new and emerging research trend, many studies have been published in the past two decades. Different ML algorithms have been adapted and used to automate software testing, however, it is not clear how research in this area has evolved in terms of what has already been investigated. Despite the inherent value of examining the nature and scope of the literature in the area, few studies have attempted to provide a general overview of how ML algorithms have contributed to efforts to automate software-testing activities. Noorian et al. [29], for instance, proposed a frame- work that can be used to classify research at the intersection of ML and software testing. Nevertheless, their classification framework is not based on a systematic review of the literature, which to some extent undermines the scope and validity of such framework.

Drawing from his personal experience, Briand [26] gives an account of the state-of-the art in ML applied to software testing by describing many applications the author was involved with over the years as well as a brief overview of other related research. Furthermore, owing to his assumption that ML has the potential to help testers cope with some long-standing software-testing problems, Briand argues that more research should be performed toward synthesizing the knowledge at the intersection of these research areas. Although evidence suggests that software testing is the subject for which a substantial number of systematic literature reviews have been carried out [30], to the best of our knowledge, there are no up-to-date, comprehensive systematic reviews or systematic mappings providing an overview of published research that combines these two research areas.

The problem statement can be still mapped to usage of machine learning techniques for automation testing. This paper provides information on using ML for automation with the use cases, challenges and software testing: outlining the most investigated topics, the strength of evidence for, and benefits and limitations of ML algorithms. The results of this systematic mapping will enable researchers to devise more effective ML-based testing approaches, since these research efforts can capitalize on the best available knowledge. In addition, given that ML is not a panacea for all software-testing issues, we conjecture that this paper is an important step to make headway in applying ML to software testing. Essentially, the results of this paper have the potential to enable practitioners and researchers to make informed decisions about which ML algorithms are best suited to their context, secondary studies as ours can be used as a starting point for further research. Another contribution of our paper is the identification of research gaps, paving the way for future research in this area.

C. Role of machine learning in automation

Basically, ML tech are trained to process information, recognize plans and designs, make and assess tests without human help. This is made conceivable with profound learning and neural systems — when a machine self-instructs dependent on the gave informational collections or information separated from an outer source, for example the web. The requirements for ML deployment require two primary approaches:

- To train the ML models to make automated tests. A few endeavors have been made in this field with changed achievement.
- To instruct AI to arrange tests, choosing self-sufficiently on what to run, what to fix, and what to dispose of.

D. Challenges in Current Automation testing

The traditional QA approach is based on the system used to test the collection of basic tasks that together shape the overall project. A tester needs to go in a way to do from the smallest elements up to the largest. Since the customers are always anxious, conventional forms of testing tend to satisfy their demands. Toward the start of a task, testing can go in corresponding with intensifying functionality, however the more complex an application turns into, the more

challenging it becomes to ensure it has full test inclusion. Consistently, QA Engineers discover a plenty of troubles and burn through a great deal of time to locate an appropriate arrangement. While including new changes, existing code that has just experienced testing may quit working. Each time the development group changes existing code, they should complete new tests. Regression testing cycles can snatch quite a while undertaking them physically can overpower QAs. This is an impressive test for some organizations. To oversee testing work processes effectively, organizations need to involve exceptionally talented experts and frequently draw in the administrations of specific accomplices who come prepared to handle the most convoluted issues and decide the best answers for every specific venture. More difficulties originate from the embodiment and objectives of automated testing, where the ML testing apparatuses are progressively finding their utilization. For groups setting up automates cases, there are the accompanying entanglements to keep an eye out for:

- *Checks just explicit cases that are picked*: If new function is included, the recently made auto-test will at present be totally effective, regardless of whether the new function doesn't work. Just research testing by humans will have the option to identify these changes. However, this regularly doesn't happen, since this time is spent on re-performing essential tests, so circumstances frequently emerge when manual testing checks application's routine as an overall thing, losing subtleties.
- *Tedious:* For automated tests to run suitably with code expansion, QA engineers need to keep tests adjusted, which ends up being progressively tangled after some time due to amassing and programming commitment.
- *Missing bugs while including new functionality:* Automated tests can run effectively; they may also fail to distinguish the code mistakes in recently included highlights. Regularly this requires manual testing, which isn't generally conceivable except if the undertaking is staffed in like manner.
- *Tend to be flaky and unstable over time as the app changes:* When actuating elongated test cases, mistakes may happen in setting up test - case steps, not in the code. In this issue, QA designers should improve a test and begin it once again. And, the code based scripting solutions will be unstable as the app keeps on adding new features and changes over time.
- *Testing architects' skills:* Test automation requires specific acquaintance and skills in producing automated test contents, implying that looking for the correct authorities can be a tedious procedure.
- *Requires human intrusion:* It's hard to automate user interface (UI) tests as it requires human intrusion and verdict.

Although not a panacea, ML developments are ready to mitigate a great part of the automated testing loads. Simultaneously, these incipient apparatuses may bring along their difficulties, which are talked about in this paper.

E. Benefits of adopting machine learning in automation

The following are the benefits of adopting machine learning techniques in automation

1. Accelerating Manual Testing and Overall Processes

Indeed, even the most progressive application advancement associations code bunches of test lines: type one line after another, "click here" and "watch that". Clearly, this strategy has numerous detriments. Making such tests confuses engineers' consideration. Complete test passing may take a few days, and several weeks. Manual testing can't maintain such a pace, regardless of how diligently it attempts. Furthermore, manual functional testing is costly, both in time and in currency.

ML will encourage such time-squandering issue for all engineers, as composing all contents and analyses a lot of informational collections turns out to be all the quicker. ML can deal with figuring out log documents thus it will spare time and upgrade accuracy in the program immensely. The potential results will give QA Engineers a total perspective on the adjustments that they should complete.

2. Mechanization the Testing Process

If the number of test are more, more is the labour cost and more expensive their support. Besides, when the application must be modified or requires changes, QA Engineers need to update the test code. Furthermore, frequently things being what they are, most of the endeavors of automation rapidly transform into clean maintenance, with a couple of changes in additional inclusion.

ML models keep on developing after changes in the code. Since the models are not completely encoded, they adjust and figure out how to discover new application functions themselves. At the point when the ML distinguishes alterations, it naturally evaluates them to choose whether this is another new feature or imperfections of another discharge. Subsequently, hard-coded test contents are delicate and require manual refinement after each adjustment in the application code and the ML models create themselves all through the whole procedure.

3. Analysis of errors

ML can discover answers to questions like how, where and when surprisingly fast or even in seconds. Along these lines, analyzers can utilize this data to choose whether coding changes are required to forestall program blunders or they basically need to apply some different methodologies. QA forms are brimming with bugs, they are the fundamental piece of this work. After all the efforts and after doing everything right there will be situations where the bug remains unnoticed. In like manner, ML in the QA situations can lead continuous investigation of bugs.

4. Decrease the Ignored Bugs Probability

The machine learning approach will offer more solid results than traditional testing does. The time expected to complete a product testing and search for a bug likewise turns diminished. The test coverage will be more and the bugs will be not being gone unnoticed. Because the issue of disregarded bugs is exceptionally assorted and bears amazingly negative outcomes. If the data management is not given enough considerations then as an outcome there will be entire bundle of unnoticed bugs which will degrade the product quality. As an overall result of the unnoticed bugs brand's notoriety will be at stake along with unsatisfied clients. But most importantly testers burn through their valuable time. But the ML approach will decrease the ignored bugs probability compared to traditional approach and increase the overall productivity.

5. Anticipating Client Requirements

Forecasting empowers enterprises to analyze customer data for a more proper understanding of the most recent products and features can be effectively explained utilizing machine learning techniques. Because finding right approach to increase the productivity and usage is essential.

6. Algorithms can eliminate poor scripts and increase time to value

The shift from Waterfall to Agile and DevOps methodologies is significant. More automation present throughout the entire pipeline of activities, including testing will lead the path to success for the devops culture.

F. Use Cases

Eliminate specific, flaky code-based test scripts: Flaky, code-based test scripts are a killer for digital quality and are often the result of poor coding practices. This reduces confidence in test automation scripts.

To know if they have flaky test scripts some indicators are:

- Test results are inconsistent from run to run or platform to platform
- Tests aren't using stable object locators
- Tests don't properly handle environment-related implications (e.g., pop-ups, interrupts, etc.)

Flaky scripts, identify the testing bottlenecks and where they are not getting value from code-based test automation. If flaky test scripts were originally developed using a code-based programming language (e.g., Java, JavaScript, Python, etc.) in an integrated development environment (IDE), can record those scripts with ML-based tools, and play them back as many times as needed across platforms. Also, it can be done from the ML tool's UI itself, or through a continuous integration (CI) server. Flaky code-based test scripts are a killer digital quality, and are often the result of poor coding practices.

Provide business testers an alternative for test automation creation: Test automation suffers from low success rates these days. In addition to flaky, code-based test scripts, there are two main reasons: Developers and test engineers are pressed for time Agile feature teams lack the skills to create automation scripts within sprints the lack of skills in Agile feature teams represents an opportunity for data scientists and business testers. These non-testers can leverage the ML-based tools and create robust test automation scripts for functional and exploratory testing through simple record and playback flows.

Increase test automation coverage: When manual testing is replaced with ML-based test automation, likely there will be increase the overall test automation coverage and reduce the risk of defects escaping into production. That's great, but still need to ensure the team works efficiently and drives value. Properly scope the ML-based test automation suite with team members to avoid duplicates and focus on problematic areas. Also, one must consider how teams will view the two methods' results. Teams must strive towards a consistent quality dashboard that shows both test automation reports in a single view so management can assess the overall product quality with ease. On average, ML-based test automation is six times faster than code-based testing, which means faster time to value.

Accelerate time to create and maintain test automation: On average, ML-based test automation is six times faster than code-based testing, which means faster time to value. What makes ML-based test automation so much faster? Code-based testing requires the developer to build the proper environment (e.g., Selenium Grid), set up the prerequisites through code and debug the code from within the IDE. This takes significant time, skills and effort — and it's not a one-time investment. As the product changes, the developer must continually update the code. On the other hand, ML-based test creation is typically a record-and-playback process with built-in self-healing algorithms. This generally does not require heavy maintenance, unless there are significant changes in the element locators or the product itself. However, ML-based tools are less mature than code-based tools. As a result, there is less flexibility and integration with other tools and frameworks.

Other than these it also includes detecting any changes in software and defining whether it's a bug or an added feature that should be tested, updating tests accordingly to the features being added, Fixing tests on the run in case a certain element is not found, Quickly detecting software changes by inspecting history logs and correlating them with the test results, Analyzing code to estimate test coverage, creating dashboards to unite and share data on tested code, current testing statuses, and test coverage, prioritizing test cases, speeding up maintenance and test runs, predicting and timely notifying about possible and code or test issues etc.

IV. CONCLUSION

ML and software testing are two broad areas of active research whose intersection has been drawing the attention of researchers. Our systematic mapping focused on making a survey of research efforts based on using ML algorithms to support software testing. Software testing approaches evolve day by day to catch up with the pace of application development techniques, becoming more complex and sophisticated with each step. As a result, fast-growing businesses often need to run ongoing, adjustable tests for their systems and products. We believe that our mapping study provides a valuable overview of the state-of-the art in ML applied to software testing, which is useful for researchers and practitioners looking to understand this research field either for the goal of leveraging or contributing to that field. Given the pressures of the high-accelerating industry, it's obvious that the new age testing is here! ML are undeniably growing to be significant elements in software testing and QA as well. And all this is for good reason. ML will advance accuracy, give enhanced revenue and lower costs for all QA processes. Henceforth, it improves competitive positioning and customer experience. Most importantly, ML helps identify bugs quicker and faster. The testers can stop worrying about losing their jobs and start focusing on making better policies. There is no reason to fear ML, instead, we should think of possible

REFERENCES

[1]. M. J. Harrold, "Testing: A roadmap," in Proc. Conf. Future Softw. Eng., 2000, pp. 61–72.
[2]. C. A. Welty and P. G. Selfridge, "Artificial intelligence and software engineering: Breaking the toy mold," Automated Softw. Eng., vol. 4, no. 3, pp. 255–270, 1997.
[3]. M. Harman, "The role of artificial intelligence in software engineering," in Proc. 1st Int. Workshop Realizing Artif. Intell. Synergies Softw. Eng., 2012, pp. 1–6.
[4]. T. Xie, "The synergy of human and artificial intelligence in software engineering," in Proc. 2nd Int. Workshop Realizing Artif. Intell. Synergies Softw. Eng., 2013, pp. 4–6.
[5]. J. Bell, Machine Learning: Hands-On for Developers and Technical Pro- fessionals. Hoboken, NJ, USA: Wiley, 2014.
[6]. D. Zhang and J. J. Tsai, "Machine learning and software engineering,"Softw. Qual. J., vol. 11, no. 2, pp. 87–119, 2003.
[7]. S. R. Vergilio, J. A. C. Maldonado, and M. Jino, "Infeasible paths in the context of data flow based testing criteria: Identification, classification and prediction," J. Brazilian Comput. Soc., vol. 12, pp. 71–86, Jun. 2006.
[8]. B. Beizer, Software Testing Techniques, 2nd ed. New York, NY, USA: Van Nostrand Reinhold Company, 1990.
[9]. H. Zhu, P. A. V. Hall, and J. H. R. May, "Software unit test coverage and adequacy," ACM Comput. Surveys, vol. 29, no. 4, pp. 366–427, 1997.
[10]. S. Rapps and E. J. Weyuker, "Data flow analysis techniques for test data selection," in Proc. 6th Int. Conf. Softw. Eng., 1982, pp. 272–278.
[11]. A. Orso and G. Rothermel, "Software testing: A research travelogue (2000–2014)," in Proc. Future Softw. Eng., 2014, pp. 117–132.
[12]. B. Lantz, Machine Learning With R, 2nd ed. Birmingham, U.K.: Packt Publishing, 2015.
[13]. M. Bowles, Machine Learning in Python: Essential Techniques for Pre- dictive Analysis. Hoboken, NJ, USA: Wiley, 2015.
[14]. M. Mohri, A. Rostamizadeh, and A. Talwalkar, Foundations of Machine Learning. Cambridge, MA, USA: MIT Press, 2012.
[15]. P. Flach, Machine Learning: The Art and Science of Algorithms That Make Sense of Data. Cambridge, U.K.: Cambridge Univ. Press, 2012.
[16]. G. James, D. Witten, T. Hastie, and R. Tibshirani, An Introduction to Statistical Learning: With Applications in R (Springer Texts in Statistics). New York, NY, USA: Springer, 2013.
[17]. P. Louridas and C. Ebert, "Machine learning," IEEE Softw., vol. 33, no. 5, pp. 110–115, Sep./Oct. 2016.
[18]. S. Shalev-Shwartz and S. Ben-David, Understanding Machine Learning: From Theory to Algorithms. Cambridge, U.K.: Cambridge Univ. Press, 2014.
[19]. C. Anderson, A. V. Mayrhauser, and R. Mraz, "On the use of neural networks to guide software testing activities," in Proc. Int. Test Conf., 1995, pp. 720–729.
[20]. H. Singh, M. Conrad, and S. Sadeghipour, "Test case design based on Z and the classification-tree method," in Proc. IEEE Int. Conf. Formal Eng. Methods, 1997, pp. 81–90.
[21]. J. Bowring, J. M. Rehg, and M. J. Harrold, "Active learning for automatic classification of software behavior," in Proc. ACM SIGSOFT Int. Symp. Softw. Testing Anal., 2004, pp. 195–205.
[22]. L. C. Briand, "Novel applications of machine learning in software testing," in Proc. 8th Int. Conf. Qual. Softw., 2008, pp. 3–10.
[23]. W. Choi, G. Necula, and K. Sen, "Guided GUI testing of android apps with minimal restart and approximate learning," in Proc. ACM SIGPLAN Int. Conf. Object Oriented Program. Syst. Lang. Appl., 2013, pp. 623–640.
[24]. J. Zhang et al., "Predictive mutation testing," in Proc. 25th Int. Symp. Softw. Testing Anal., 2016, pp. 342–353.
[25]. M. Noorian, E. Bagheri, and W. Du, "Machine learning-based software testing: Towards a classification framework," in Proc. Int. Conf. Softw. Eng. Knowl. Eng., 2011, pp. 225–229.
[26]. K. Petersen, S. Vakkalanka, and L. Kuzniarz, "Guidelines for conduct- ing systematic mapping studies in software engineering: An update," Inf. Softw. Technol., vol. 64, pp. 1–18, 2015.
[27]. B. A. Kitchenham, D. Budgen, and P. Brereton, Evidence-Based Software Engineering and Systematic Reviews. London, U.K.: Chapman and Hall/, 2015.

[28]. B. A. Kitchenham, D. Budgen, and O. P. Brereton, “Using mapping studies as the basis for further research – A participant-observer case study,” Inf. Softw. Technol., vol. 53, no. 6, pp. 638–651, 2011.
[29]. V. R. Basili, G. Caldiera, and H. D. Rombach, “The goal question metric approach, ” in Encyclopedia of Software Engineering. Hoboken, NJ, USA: Wiley, 1994.
[30]. H. Zhang, M. A. Babar, and P. Tell, “Identifying relevant studies in soft- ware engineering,” Inf. Softw. Technol., vol. 53, no. 6, pp. 625–637, 2011.
[31]. C. Wohlin, “Guidelines for snowballing in systematic literature studies and a replication in software engineering,” in Proc. 18th Int. Conf. Eval. Assessment Softw. Eng., 2014, paper 38.
[32]. Vinicius H. S. Durelli, Rafael S. Durelli, Simone S. Borges, Andre T. Endo et al. "Machine Learning Applied to Software Testing: A Systematic Mapping Study", IEEE Transactionsnon Reliability, 2019