J. Integr. Sci. Technol. 2023, 11(3), 517 Article

Journal of Integrated
SCIENCE & TECHNOLOGY

An analytical study on testing metrics for software applications

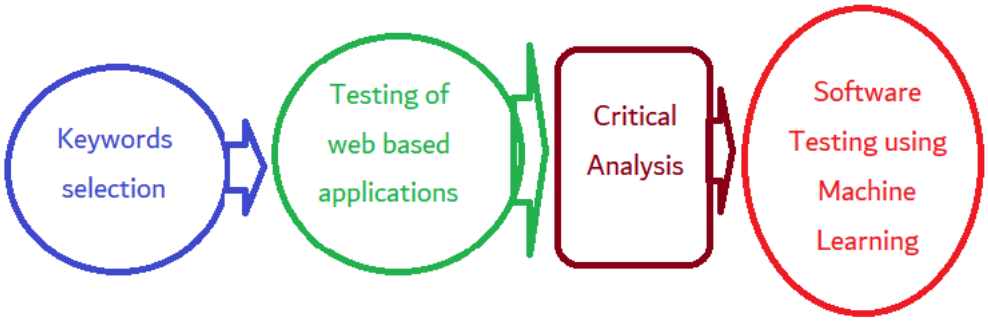
Kamal Kant Sharma[1,2*], Amit Sinha[3], Arun Sharma[4]

[1]KIET Group of Institutions, Ghaziabad, India. [2]Dr. A.P.J. Abdul Kalam Technical University, Lucknow, India. [3]ABES Engineering College, Ghaziabad, India. [4]Indira Gandhi Delhi Technical University for Women (IGDTUW), Delhi, India

Submitted on: 08-Oct-2022; Accepted and Published on: 20-Jan-2023

ABSTRACT

In day-to-day life, software applications have acquired an important part. Some of these applications are open source and some are paid ones. Majority of Users mainly search for free, error-free apps that can meet their requirements. So, to make error free applications, proper testing is required which requires more time and efforts. Similar to desktop applications, black box or white box testing can be applied to mobile applications too. The main objective of this paper is to review on different testing methods used for testing mobile applications, web applications or artificial intelligence applications. The paper reviewed and analyzed the recent contributions of researchers using machine learning approach for testing software applications and identified the limitations and future research scope in this field.



Keywords: Testing, Machine Learning, Mobile Applications, Software.

INTRODUCTION

The phrase "software metrics" is a somewhat deceptive aggregate word used to cover a variety of operations associated with evaluation in software engineering. These activities vary from creating figures that describe software code characteristics (the basic software metric) to algorithms that assist to anticipate software material requirements and system functionality.[1] The topic also encompasses the quantifiable components of quality management that include tasks such as flaw tracking and reporting throughout design and testing. Given the repercussions of software faults in terms of life, money, and time delays, the importance of quality software is no longer a bonus, but rather a requirement. Without any doubt, the quality of software can make or break a company. Unfortunately, most firms not only fail to provide their customers with a high-quality item, but also struggle to recognize the qualities of a great product.[2]

*Corresponding author: Kamal Kant Sharma, KIET Group of Institutions, Ghaziabad, India
Email: kantkkiet@gmail.com

Cite as: *J. Integr. Sci. Technol., 2023, 11(3), 517.*
URN:NBN:sciencein.jist.2023.v11.517

 ISSN: 2321-4635
http://pubs.thesciencein.org/jist

Traditional software metrics often used to assess product qualities including size, intricacy, and efficiency, however, these are being replaced by various features implicit in object-orientation, like encapsulation, derivation, and flexibility.[3] This transition resulted in the development of several metrics offered by various scholars and practitioners to quantify object-oriented characteristics. The majority of metrics accessible for object-oriented software analysis is often employed in the later stages of the implementation phase and collects the information collected from software implementation process. Such measurements offer an indicator of quality too late to enhance the object before it is completed. It is therefore relevant that a combination of object-orientated metrics may be employed to assess all elements of object-oriented programming. The signaling and forecasting of performance as soon as feasible in the System Development Life Cycle (SDLC) is required since the cost effect of change and enhancement increases considerably with each repetition of SDLC.[4-11]

The potential to protect all elements of quality variables and key strategies, to reflect various elements of the program under assessment, to get the identical significance for the same system for different people at different times, to use the least variety of measurements, to have empirical evidence, and to operate without malfunction are recognized as central characteristics of the object-

oriented metrics. In addition to these key features, it is believed that the intended metrics should have the ability to calculate different features of reutilization, to decrease re - work after execution, to minimize testing and maintenance costs, and to guarantee that design process have favorable internal characteristics that will contribute to the development of quality. These metrics give methods for evaluation of software quality, and their usage in earlier stages of software development can assist businesses in swiftly reviewing a big software development at a minimal cost.[12-15]

But how do we determine which indicators are appropriate for tracking essential quality traits like error propensity, effort, efficiency, or the number of maintenance adjustments? An extensive analysis of real-world systems only yields meaningful results. In subsequent papers, the authors offer a more comprehensive treatment of their candidate metrics. Prior to beginning any measuring activity, we must first determine the characteristic to be assessed. Such a characteristic must be meaningful to someone associated with developing, such as an architect, developer, administrator, user, and so on.[16,17]

The characteristic may not be noteworthy in and of itself, but it may function as a predictor variables in the indirect assessment of the other characteristic or in a particular prediction model. The "satisfactory" empirical relationship systems, that is, an empirical connection system that captures all widely recognized instinctive concepts about the characteristic under discussion, must be created. In what follows, we use a specific coupling metric to show the implications of not rigorously following to this criterion.[18] Coupling is a well-known internal product feature that has been researched since the emergence of structured programming and, more subsequently, in the scope of object-orientation. The concept of coupling varies slightly from the traditional one under the object-oriented model, but the essential principles stay same. Software metrics are the deciding elements for measuring quality of software in terms of complexity, performance, etc.[19-30] Metrics are critical in developing acceptable techniques to evaluate the technology.[28] Further we discuss about the related research in the field of testing metrics, the theory acquired by several researches in the recent years. And how that will be helpful in upcoming researches and experiments.

RELATED WORK

In this paper, software testing of different software applications are presented using machine learning approach. For this a systematic critical analysis is presented, as presented in figure 1.

Testing for Mobile Applications

Bokingkito Jr et al.[28] identified how individuals, or fisher-folk, react to a mobile water quality assessment application in respect of its efficacy, efficiency, and contentment, which all add up to its usefulness. The outcomes were acquired by conducting feasibility analysis using a modified Goal Question Metric approach, which included tests and questionnaires. According to the assessment outcomes, the Real-Time Water Quality Monitoring app attains higher degree of success in respect of efficacy and efficiency, and consumers are happy with it. There have been, yet, a few issues that required to be solved.

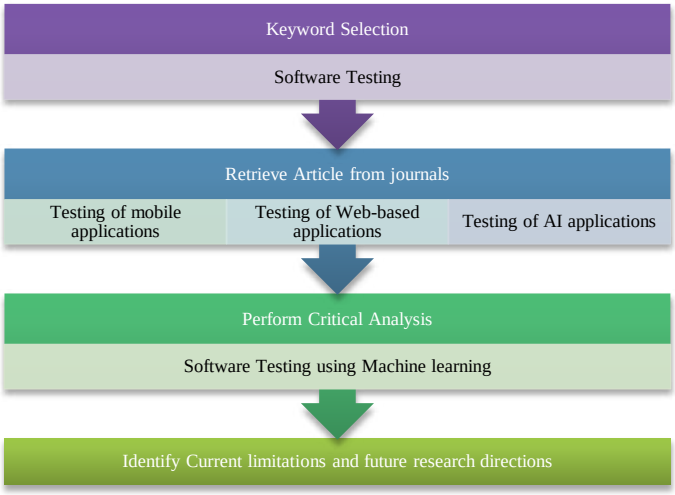


Figure 1. Approach for Systematic Meta-Analysis

Liu et al.[29] described an unique approach for evaluating the reaction duration of mobile applications on various smart phones by integrating network protocol analysis with data fusion. They invited several individuals to gather information on their smart phones to make the information acquired through the smartphone application more dependable and authentic. The network protocol evaluation approach was then utilised to determine the respond duration of the smartphone application on a cell phone. Following that, researchers used data fusion technique with the rank-score characteristic function to assess the reaction speed of mobile applications on various smart phones. Tests were carried out to assess the method on three different kinds of mobile applications. The findings demonstrated that the suggested technique is capable of evaluating the reaction duration of mobile applications with minimal price.

Zhauniarovich et al.[30] created BBOXTESTER, a technology that can provide code coverage information and consistent coverage metrics in assessment without requiring the source code. Security professionals may autonomously run apps using current state-of-the-art techniques and utilise the findings of their framework to determine whether or not the security-critical code was included by the testing. Researchers reported on the designing and execution of BBOXTESTER in this study and evaluate its reliability and efficacy.

Testing of Web Software Applications

Gao et al.[31] provided a simple, but efficient, model-based automation testing strategy for achieving high code coverage while staying under a time constraint by testing with extended activity patterns. They implemented this method as an open-source framework called LJS, and they run thorough trials on 21 publically accessible standards. LJS can reach 86.5 percent line coverage in 10 minutes on average. On real-world big Web apps, LJS's coverage is 11–19% greater than that of JSDEP, a cutting-edge breadth-first search-based autonomous screening technique augmented with partial order reductions. The empirical outcomes support the notion that prolonged testing periods can provide more code coverage in JavaScript Web app assessment.

Suguna Mallika et al.[32] presented mutation operators, a general set for testing web applications. The entire set of the operators is provided in this research as an extensive regression testing process to mitigate web app vulnerabilities. Small and medium-sized businesses can rely on the regression testing process provided for consumer acquisition while reducing money on screening costs. The testing team guarantees a strong web application that is resistant to unauthorized access.

Testing of AI Software Applications

Wan et al.[33] proposed a novel assessment technique for applications that employs cognitive ML APIs, as described in this article. Keeper creates a pseudo-inverse function for every ML API that empirically overturns the associated task effectively (e.g., a photograph search tool pseudo-reverses the image-classification API), and then encompasses all such pseudo-inverse operations into a representational application server to start generating appropriate photograph inputs and judge output accurateness. Keeper tries to improve how ML APIs are utilized in applications to reduce misbehaviour after it has been uncovered. Keeper dramatically increases branch coverage while detecting numerous earlier undisclosed problems, according to their testing on a number of open-source apps.

Haldar et al.[34] made the first step by developing AITEST, an assessment platform for transformational qualities including impartiality and durability in tabulated, time-series, and text categorization framewResearchers extended the AITEST tool's capabilities in this study to incorporate assessment approaches for Photograph and Speech-to-text models, as well as assessment of ability to understand for tabular models. AITEST is now a complete tool for evaluating Artificial intelligence systems because of these new features.

With the extensive use of Ai techniques for critical decision-making, assuring their dependability remains a significant concern. Aggarwal et al.[35] provided an end-to-end modular approach for evaluating Artificial Intelligence systems in this study, which generates autonomous tests for numerous paradigms which including textual, tabulated, and time-series information, as well as for several attributes like as precision, impartiality, and durability. This technology has also been utilized to assess commercial Artificial intelligence systems and has shown to be quite successful in revealing flaws in those methods.

Deep learning (DL) has made incredible development over the last decades and is now extensively used in a variety of commercial areas. Nonetheless, the resilience of DL algorithms has lately become a major challenge, with small variations on the input causing the DL to fail. These resilience difficulties may have serious impacts when a DL framework is implemented into safety-critical implementations, and they may impede the real-world implementation of DL frameworks. Assessment approaches allow for the analysis of a DL system's durability and the identifying of susceptible issues at a beginning period. The major constraint in evaluating a DL framework is due to the great dimensionality of its inputs and the huge internal latent characteristic area, which renders evaluating every phase almost unfeasible. Combinatorial testing (CT) is an excellent screening approach for conventional software to optimize screening exploratory efforts with problem identification capability. Critical analysis of these works is presented in table 1.

Table 1. Critical Analysis of Approaches Used

	Technique	Advantages	Disadvantages
[30]	BBOXTESTER	BBOXTESTER enables a tester to choose when to begin and end the gathering of coverage data. It has an unusual logging mechanism and may thus be utilized to find problems. They provided BBOXTESTER open source3 in order to stimulate more study in this field.	BBOXTESTER only analyzes code coverage of Dalvik code; it does not compute coverage of native code. The practice of converting from Dalvik bytecode to Java bytecode and then compiling it backward to Dalvik might generate extra faults, prohibiting apps from being correctly instrumented.
[29]	SRBM	On most databases, SRBM outperforms other approaches in respect of clustering evaluations metrics, specifically in the context of zeroday malware identification.	RBM's functioning can be improved by SRBM, although on some databases, it still fails miserably.
[8]	reinforcement learning	Lower the amount of time it takes to examine the particular app under examination. Over most sets of data, the suggested technique beats the existing techniques.	Excessive reinforce ment training might result in an overabundance of situations, lowering the quality of the findings. For basic tasks, this technique is not the best option.
[34]	Keeper	It detects errors that cause software to malfunction, reduce inference effectiveness, or result in useless code.	Keeper's efforts to identify and resolve problems are uncertain by nature.
[35]	AITEST	Contains a wide range of examination methods over many disciplines. Speech-to-text systems now have new features like as understandability and impartial evaluation, as well as the execution of several well-known attributes.	This only enables black-box assessment; videos, multi-modal inputs, and model configurations are not supported.

SOFTWARE TESTING USING MACHINE LEARNING

Yao et al.[1] proposed a methodology based on genetic algorithm for improvement of quality of software before its usage. The algorithm was based on randomness of parameters for software testing. The usage of genetic algorithm is time consuming. In that work, author applied genetic algorithm for testing software with randomness. The proposed method was tested on 12 test programs with genetic algorithm. The result accuracy was performed on three stating criteria's such as statement coverage, branch

coverage and path coverage testing and achieved accuracy of 95%, 94% and 93% respectively.

Yan et al.[2] proposed a feature-based euclidean distance (FED) to evaluated the failure and non-failure of software for this deep learning was used. Random testing was adopted to evaluate the performance. The detection rate was evaluated as 62.74%.

Samet[3] applied machine learning approach for testing failure and success of mobile applications. The algorithm covered data security of application as a major concern. Continuous Proof of Presence was used for testing purpose. But this model was not adaptable for web/desktop applications.

Yan et al.[4] proposed a toolkit for usability testing of mobile apps that automatically collects information from user interface (UI) events. The model accurately analyzed the usability of applications and are only dedicated towards usability testing, other aspects of testing was not covered. The model is not adaptable for web/desktop applications.

Costa et al.[5] proposed a pattern Based GUI testing for mobile applications that iincreases the systematization, reusability of the model. The author presented a study on mobile test application but are not adaptable for web/desktop applications.

Salihu et al.[6] proposed a hybrid model termed as AMOGA for testing of mobile application. Event list were with crawling technique that finds the association among them. Rosenfeld et al.[7] proposed a model for functional testing of mobile applications using machine learning. The approach identified the common behaviour among software UI screen and also distinguished general scripts that has been instantiated and reused. Brito e Abreu & Melo[8] described the findings of a study that looked at the impact of the Object-Oriented paradigm on software dependability attributes. To track the implementation of OO design methodologies, MOOD, a set of OO design metrics, was implemented. The findings of a comprehensive study of software maintainability prediction and metrics are presented by Riaz et al.[9] Their research focused on the software qualitative attribute of manageability rather than the software management procedure. Their findings imply that there is minimal evidence that software maintainability prediction approaches and frameworks are useful. For verifying maintainability prediction models, the usage of prediction methodologies and models, reliability measures, and cross-validation procedures was found to be uncommon; and the most widely used maintainability metric used an ordinal scale and was dependent on specialist judgement.

Barkmann et al.[10] developed instruments for metrics analysis of a huge quantity of software programs to aid in the solution. Furthermore, software quality metric validation should emphasize on meaningful measures, rather than linked metrics, which do not require to be evaluated separately. Challagulla et al.[11] reported 4 distinct real-time software defect data sets to examine several predictive models. Their findings also reveal that "size" and "complexity" metrics aren't enough to effectively forecast real-time software faults. In comparison to other models, the outcomes suggest that combining 1R and Instance-based Learning with the Consistency based Subset Assessment methodology delivers a considerably superior coherence in accuracy forecasting.

Burton-Jones et al.[12] recommended a set of metrices to evaluate the reliability of an ontology. The metrics, which are based on semiotic theory. The metrics are operationalized and implemented in a prototype instrument called the Ontology Auditor. The study makes a theoretical influence by providing a mechanism that developers can use to create high-quality ontologies and applications may use to select relevant ontologies for a given assignment. The findings demonstrated that there are a lot of places where developers 'ontologies might be improved.

El-Emam[13] looked at current object-oriented metrics, their theory, and the empirical evidence that backs them up. The findings thus far can be used to develop concrete performance assurance criteria for object-oriented programmes. The metrics that measure the various types of export and import couplings appear to be the most relevant to collect. The majority of these metrics have the benefit of being able to be gathered early in the design process, allowing for early performance measurement. Assign your most capable employees to courses with high coupling metrics. This entails building a logistic regression system employing the coupling measures mentioned above (and a measure of size). This framework would forecast the likelihood of a mistake in every session.

Software metrics have a long history, almost as long as software engineering itself. However, despite substantial studies and writing on the subject, industrial practise has remained largely unchanged. This is concerning, considering that the primary motivation for utilising metrics is to better the managerial and technical aspects of software engineering decision-making. The main issue is that such measurements are often used in isolation. We propose that by taking a less isolationist perspective, it is possible to deliver genuinely enhanced management decision support systems based on such basic criteria. In regards of industrial penetration, the vast majority of academic metrics research has failed. Given the subject's fundamentally "adapted" nature, this is an especially damning criticism.[14]

Fenton & Neil[15] performed a comprehensive analysis of existing software failure forecasting studies, focusing on metrics, techniques, and datasets. Since 2005, the use of public datasets has expanded dramatically, while the use of machine learning methods has increased little, according to the review findings. Furthermore, method-level measures remain the most prevalent metrics in the defect detection research field, while machine learning algorithms remain the most common fault prediction methods. To improve fault predictors, researchers working in the field of software failure prediction should continue to employ machine learning methods.

Catal and Diri[16] presented the review on fault prediction of software and proved that machine learning algorithms is better solution for fault prediction. Khan et al.[17] introduced Weighted Class Complexity (WCC) as an object-oriented design measure in this work. It is discussed from an assessment theory perspective that takes the known object oriented functionalities that the metrics was designed to evaluate, such as encapsulation, inheritance and coupling anpolymorphism, and the quality elements effectiveness, ambiguity and understandability, as well as the quality variables effectiveness, complexity, understandable, reusability and manageability into consideration. The WCC metric is used to

examine real data from eight different application fields to support this theoretical validity. According to the findings, the recommended metric can be used early in the development process to evaluate the overall effectiveness of an object-oriented software system. The developer can use this in the early stages of implementation to resolve faults, decrease irregularities and non-conformance to standards, and avoid unnecessary complexity. If this metric is used early in the process, it can help ensure that the assessment and design have favourable internal aspects, leading to the creation of an excellent product.

Dalcher & Raffo[18] As software measurement becomes more important, innovative measurement technologies are being developed. The concept of OOPS were used in this work. The goal of this paper was to look into the connections among established design metrics and the likelihood of detecting faults in classes. The research presented here is a replication of a similar research. The goal is to produce empirical evidence so that strong findings can be drawn from multiple investigations. The findings of this study reveal that multiple measurements in a metric set capture the identical characteristics, implying that they are relied on similar principles and provide duplicate information. It is demonstrated that by combining a selection of measurements, prediction models for identifying incorrect classes may be created. The anticipated model demonstrates that import coupling and size metrics are substantially connected to fault proneness, correlating with earlier research. Clevenger & Haymaker[19] have characterized design process in their study using three dimensions: challenge, strategy, and exploration. The study reported a paradigm for defining the elements, spaces of performance-based design with precision and created a set of metrics that may be used to track all aspects of the design procedure. Ultimately, the study illustrated the framework's and metrics' use by using them to compare two different tactics on a task. Authors reported that the structure and metrics make it easier to compare and contrast design procedures. The metrics' strengths comprise the capacity to evaluate variations in issues that were previously impossible to quantify employing traditional point-based design approaches. The most essential and contentious indicator is Alternative Space Flexibility (ASF). The measure demonstrates that extensive analysis is the least creative method in the research papers. The conflict among creativity and systematic search, as well as the link among creativity and breakthrough achievement, has long been recognized by experts.

Shatnawi[20] With scarce resources, software engineers want metrics analysis instruments to analyze programme performance, such as module fault-proneness. There are numerous software measures that can be used to assess performance. Not all measurements, although, are significantly linked to defects. The employment of ROC to establish thresholds for four indicators was validated in this study (WMC, CBO, RFC and LCOM). After sampling the data, the ROC outcomes are not significantly varied from before sampling. In most datasets, the ROC analysis uses the same measures (WMC, CBO, and RFC), whereas other methodologies select metrics differently.

Saxena et al.[21] utilised testing metrices for prognostics in a range of sectors, which include medicine, nuclear, automotive, aerospace, and electronics, are reviewed in this work. Other fields that require prediction-related projects, such as weather and banking, are also considered. The distinctions and commonalities between these domains and health maintenance were examined in order to determine which quality assessment techniques could be borrowed and which could not. Furthermore, these indicators have been classified in a variety of ways that can help you choose the right subset for your needs. Certain key prognostic notions have been described employing a notational paradigm that allows for a consistent interpretation of various measures. Finally, a set of criteria has been proposed for evaluating essential aspects of RUL forecasts before they are used in real-world applications.

Hitz & Montazeri[22] employed the object coupling measure (CBO) as an approach to establish relationship among software metrices and identified the defects. Suresh et al.[23] estimated the quantitative metrices of the extent to which an organization, a component, or a method demonstrates a given trait, are the best approach to convey assessment in software engineering. Analysts, designers, programmers, and testers all want software metrics to help them better understand and plan their work. Traditional and object-oriented methodologies will be used in this study to evaluate ATM software using a subset of metrics. Classical measures such as cyclomatic complexity, size, and comment percent are used to calculate the program's complexity. An object-oriented metric suite, such as the metric suite, is used to compute system dependability. The ATM software's complexity and dependability can be determined by analyzing the metric values in a real-world application. Abaei & Selamat[24] proposed a machine learning model with feature selection strategies to forecast the failure of applications. The result was evaluated on probability of detection, probability of failure, and AUC. The author identified the important features and metrices that are suitable for enhancement of forecasting outcomes. Lanus et al.[25] addressed metrics that can be used to describe classify results and define the domain (operational envelope) in which machine learning algorithms can be anticipated to perform well. Future research is intended to investigate the utility of these metrics across a variety of ML domains, to experiment the hypothesis that models trained on source sets with smaller SDCCMt distances to the target will function effectively in the target environment, to investigate the effect of label centrism, and to determine how to choose a “good” interaction size t. Antsaklis[26] reported a fundamental concept of autonomous systems, which obviously proceeds to the construction of metrics to assess a system's level of autonomy. This definition is predicated on a system's capability to accomplish objectives in the face of uncertainty, and it excludes the means by which the targets are attained, such as sensing and responses. Avazpour et al.[27] looked at a variety of assessment metrics and measures, as well as several methodologies for assessing recommendation systems. The metrics described in this article are organized into sixteen categories, such as correctness, novelty, and coverage. They looked at these measures in terms of the dimensions they belong to. The following is a quick summary of ways to thorough assessment considering sets of recommendation system aspects and associated metrics. In addition, authors made recommendations for important upcoming studies and practise areas.

Table 2. Noteworthy Contributions of Researchers

Methods	Discussions
Genetic Algorithm [1]	Dedicated for improvement of quality of software before its usage. Based on randomness of parameters for software testing. Usage of genetic algorithm is time consuming.
Random Method [2]	Feature-based Euclidean Distance (FED) as the distance metric that can be used to measure the difference between failure-causing inputs and non-failure-causing inputs. This approach was only dedicated for deep learning programs.
Touch Metric [3]	Covered data security of application as major concern. Continuous Proof of Presence was used for testing purpose. Not adaptable for web/desktop applications.
Usability Testing [4]	Automatically collects information from user interface (UI) events. Accurately analyzed the usability of applications. Only dedicated towards usability testing, other aspects of testing was not covered. Not adaptable for web/desktop applications
User Interface Test Patterns [5]	Increases the systematization, reusability of the model. Presented a study on mobile test application. Not adaptable for web/desktop applications.

TYPES OF TESTING

Combinatorial Testing

Combinatorial testing is a type of evaluation in which several permutations of input variables are employed to measure a software product. The goal is to verify that the products is bug-free and it can control multiple input configuration pairings or situations. The pairwise system testing, that also includes defining all pairs of input variable values, is among the most frequently used combinatorial methodologies. The essential discovery of combinatorial testing (CT) is whether or not a variable allocation causes a problem, but rather it is the connections among various variables that cause software defects.[12] A CT test suite, also known as a t-way coverage arrays, is intended to evaluate variable effects up to "strength' by only exploring possibilities among a specific amount of t variables. The variable interactions in CT may be described as a t-way mixture, that is, a combo of t parameter values.

Random Testing

Random testing, often called as monkey testing, is a type of operational black box testing being used when there is insufficient time to develop and execute tests. Random testing (RT) establishes test inputs at unexpected times rather than purposefully taking samples test scenarios to target specific types of errors. This method is pretty straightforward and quick to set up, but it has proved useful, and this is one of the very few test methods whose fault diagnosis capabilities has indeed been theoretically investigated. Random testing is used when faults are not found at regular intervals. The system's dependability and efficiency are tested using random input. This method saves a lot of time in comparison to genuine testing efforts. Random inputs are found for comparison with the system. Testing input are processed separately of the test area. These randomized inputs are used to execute tests. Keep track of the results and compare them to the predicted outcomes. Reproduce the problem and report any flaws, then repair and recheck.

Adaptive Random Testing

The goal of adaptive random testing is to disperse test cases more equally throughout the feature space. It is founded on the premise that for non-point failing scenarios, an equal distribution of test scenarios is more likely to recognize errors with less test cases than standard random testing. Considering the fact that fault-causing inputs are frequently grouped into part of the regional, adaptive random testing (ART) was suggested to enhance random testing by distributing randomly chosen test scenarios as equally as feasible across the entire state space. Divergence test is performed in ART.

The fault detection capabilities and computational cost of CT, RT, and ART when tested under varying proportions of accessible parameters and restrictions in the model, as well as fault failure rates. CT is strongly recommended for use since it performed no worse than the other two approaches in 98% of test situations evaluated, and it improves when the defect is difficult to identify and all limitations are accurately recognized in the models.

RESEARCH CHALLENGES, LIMITATIONS, AND FUTURE SCOPE

Following problems are identified:

- From the literature review presented above, it is observed that significant work has been done in the area of testing mobile application. There is a scope of research in metrics like security and performance in order to understand the complexity of testing mobile application. Following problems are identified in this research:
- Many applications cannot handle high concurrent pressure and perform stable operation.
- Concurrency, fault tolerance, stability, time, cost, etc. are important factors that needed to be determined.

A cost-effective mobile application testing method can be achieved by careful consideration of the target devices, connectivity possibilities, and testing technologies that maximise automation. Automating as much of the testing as possible is a great strategy to speed up the process and save money in the long run. When selecting automation tools, consideration should be given to factors such as mobile platform support, script reusability, and total cost of ownership. The efficiency of mobile application development can be improved by using a specific type of measure. More people will be able to use the application effectively as a result.

Software testing will increasingly rely on machine learning,[36] according to current trends. One of the key drivers of change in the future is machine learning. It has already begun implementing some striking improvements in the creation and utilization of apps. The market for artificial intelligence is rapidly expanding, which suggests that machine learning is becoming more and more prevalent in the IT sector. However, this trend won't make programming languages and their frameworks obsolete. The following may be checked and tested using machine learning, such as the optimization of the test suite to identify excessive or the

opposite. i.e., certain code checks. based on prior inspections, prediction of the essential test configurations. After that, automated identification checks should be able to be performed. logging analytics, Along with determining the high risk application state for the regression test ranking and Analytics for defects predictions.

CONCLUSION

With careful consideration of target devices, connectivity choices, and testing technologies that maximise automation, a cost-effective mobile application testing procedure can be achieved. Automating as much of the testing as possible not only speeds up the process but also lowers overall testing costs. In this work, a variety of machine learning algorithms are examined, and potential future research problems are indicated.

CONFLICT OF INTEREST

Authors do not have any conflict of interest for this work.

REFERENCES

1. X. Yao, D. Gong, B. Li, X. Dang, G. Zhang. Testing Method for Software with Randomness Using Genetic Algorithm. *IEEE Access* **2020**, 8, 61999–62010.
2. M. Yan, L. Wang, A. Fei. ARTDL: Adaptive Random Testing for Deep Learning Systems. *IEEE Access* **2020**, 8, 3055–3064.
3. S. Samet, M.T. Ishraque, M. Ghadamyari, et al. TouchMetric: a machine learning based continuous authentication feature testing mobile application. *Int. J. Inf. Technol. 2019 114* **2019**, 11 (4), 625–631.
4. X. Ma, B. Yan, G. Chen, et al. Design and implementation of a toolkit for usability testing of mobile apps. *Mob. Networks Appl.* **2013**, 18 (1), 81–97.
5. P. Costa, A.C.R. Paiva, M. Nabuco. Pattern based GUI testing for mobile applications. *Proc. - 2014 9th Int. Conf. Qual. Inf. Commun. Technol. QUATIC 2014* **2014**, 66–74.
6. I.A, R. Ibrahim, B.S. Ahmed, K.Z. Zamli, A. Usman. AMOGA: A Static-Dynamic Model Generation Strategy for Mobile Apps Testing. *IEEE Access* **2019**, 7, 17158–17173.
7. A. Rosenfeld, O. Kardashov, O. Zang. Automation of android applications functional testing using machine learning activities classification. In Proceedings of the 5th International Conference on Mobile Software Engineering and Systems, **2018**, pp. 122-132.
8. F. Brito e Abreu, W. Melo. Evaluating the impact of object-oriented design on software quality. *Int. Softw. Metrics Symp. Proc.* **1996**, 90–99.
9. M. Riaz, E. Mendes, E. Tempero. A systematic review of software maintainability prediction and metrics. *2009 3rd Int. Symp. Empir. Softw. Eng. Meas. ESEM 2009* **2009**, 367–377.
10. H. Barkmann, R. Lincke, W. Löwe. Quantitative evaluation of software quality metrics in open-source projects. *Proc. - Int. Conf. Adv. Inf. Netw. Appl. AINA* **2009**, 1067–1072.
11. V.U.B. Challagulla, F.B. Bastani, I.L. Yen, R.A. Paul. Empirical assessment of machine learning based software defect prediction techniques. *Proc. - Int. Work. Object-Oriented Real-Time Dependable Syst. WORDS* **2005**, 263–270.
12. A. Burton-Jones, V.C. Storey, V. Sugumaran, P. Ahluwalia. A semiotic metrics suite for assessing the quality of ontologies. *Data Knowl. Eng.* **2005**, 55 (1), 84–102.
13. K. El-Emam. Object-Oriented Metrics: A Review of Theory and Practice. *Adv. Softw. Eng.* **2002**, 23–50.
14. V. Côté, P. Bourque, S. Oligny, N. Rivard. Software metrics: An overview of recent results. *J. Syst. Softw.* **1988**, 8 (2), 121–131.
15. N.E. Fenton, M. Neil. Software metrics. *J. Syst. Softw.* **1999**, 47 (2), 149–157.
16. C. Catal, B. Diri. A systematic review of software fault prediction studies. *Expert Syst. Appl.* **2009**, 36 (4), 7346–7354.
17. R.A. Khan, K. Mustafa, S.I. Ahson. An empirical validation of object-oriented design quality metrics. *J. King Saud Univ. - Comput. Inf. Sci.* **2007**, 19, 1–16.
18. D. Dalcher, D. Raffo. Software process: The end of an Era. *Softw. Process Improv. Pract.* **2009**, 14 (6), 303–304.
19. C.M. Clevenger, J. Haymaker. Metrics to assess design guidance. *Des. Stud.* **2011**, 32 (5), 431–456.
20. R. Shatnawi. The application of ROC analysis in threshold identification, data imbalance and metrics selection for software fault prediction. *Innov. Syst. Softw. Eng. 2017 132* **2017**, 13 (2), 201–217.
21. A. Saxena, J. Celaya, E. Balaban, et al. Metrics for evaluating performance of prognostic techniques. *2008 Int. Conf. Progn. Heal. Manag. PHM 2008* **2008**.
22. M. Hitz, B. Montazeri. Chidamber and kemerer's metrics suite: A measurement theory perspective. *IEEE Trans. Softw. Eng.* **1996**, 22 (4), 267–271.
23. Y. Suresh, J. Pati, S.K. Rath. Effectiveness of Software Metrics for Object-oriented System. *Procedia Technol.* **2012**, 6, 420–427.
24. G. Abaei, A. Selamat, G. Abaei. A Selamat. A survey on software fault detection based on different prediction approaches. *Vietnam J. Comput. Sci. 2013 12* **2013**, 1 (2), 79–95.
25. E. Lanus, L.J. Freeman, D. Richard Kuhn, R.N. Kacker. Combinatorial testing metrics for machine learning. *Proc. - 2021 IEEE 14th Int. Conf. Softw. Testing, Verif. Valid. Work. ICSTW 2021* **2021**, 81–84.
26. P. Antsaklis. Autonomy and metrics of autonomy. *Annu. Rev. Control* **2020**, 49, 15–26.
27. I. Avazpour, T. Pitakrat, L. Grunske, J. Grundy. Dimensions and metrics for evaluating recommendation systems. *Recomm. Syst. Softw. Eng.* **2014**, 245–273.
28. P.B. Bokingkito, L.T. Caparida. Usability evaluation of a real-time water quality monitoring mobile application. *Procedia Comput. Sci.* **2022**, 197, 642–649.
29. Y. Li, Y. Feng, R. Hao, et al. Classifying crowdsourced mobile test reports with image features: An empirical study. *J. Syst. Softw.* **2022**, 184.
30. Y. Zhauniarovich, A. Philippov, O. Gadyatskaya, B. Crispo, F. Massacci. Towards black box testing of android apps. *Proc. - 10th Int. Conf. Availability, Reliab. Secur. ARES 2015* **2015**, 501–510.
31. P. Gao, Y. Xu, F. Song, T. Chen. Model-based automated testing of JavaScript Web applications via longer test sequences. *Front. Comput. Sci. 2022 163* **2021**, 16 (3), 1–14.
32. S. Suguna Mallika, D. Rajya Lakshmi. Mutation testing and web applications—a test driven development approach for web applications built with java script. *Intell. Syst. Ref. Libr.* **2022**, 210, 243–259.
33. C. Wan, S. Liu, S. Xie, et al. Automated testing of software that uses machine learning APIs. *Proc. - Int. Conf. Softw. Eng.* **2022**, 2022-May, 212–224.
34. S. Haldar, D. Vijaykeerthy, D. Saha. Automated Testing of AI Models. **2021**, *arXiv preprint arXiv:2110.03320*.
35. A. Aggarwal, P. Lohia, S. Nagar, K. Dey, D. Saha. Black box fairness testing of machine learning models. *ESEC/FSE 2019 - Proc. 2019 27th ACM Jt. Meet. Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.* **2019**, 625–635.
36. N. Yadav, V. Yadav. Software reliability prediction and optimization using machine learning algorithms: A review. *J. Integr. Sci. Technol.* **2023**, 11 (1), 457.