JOURNAL OF SCIENCE TECHNOLOGY AND EDUCATION 11(1), MARCH, 2023
ISSN: 2277-0011; Journal homepage: www.atbuftejoste.net

A Review on the New Trend in Regression Test Case Prioritization

Hassan Abubakar[1], Fatima Umar Zambuk[1], Usman Mahmud Ahmed[2], Abdulsalam Ya'u Gital[1]
[1]*Faculty of Science,*
Department of Mathematical Sciences,
Abubakar Tafawa Balewa University, Bauchi, Nigeria
[2]*Faculty of Management Science,*
Department of Management & Information Technology,
Abubakar Tafawa Balewa University, Bauchi, Nigeria

ABSTRACT
The fourth industrial revolution further triggered frequent software evolution, necessitating retesting the system to assure its validity. As a result, regression testing ensures that this occurs. Test Case Prioritization, which organizes test cases based on a specific criterion to be tested according to available resources, ensuring that the most critical test cases are tested first, is one of the most prominent fields of regression testing study. To investigate the research area's trend, 250 scientific articles from 2002 to 2021 were chronologically analyzed in this work. This study looks at different forms of test case prioritization and describes some of the strategies used and the problem that each study raises. It will provide an overview of some major issues currently facing test case prioritization and some suggestions for future research areas to investigate based on the current trend in the field.

ARTICLE INFO
Article History
Received: March, 2023
Received in revised form: May, 2023
Accepted: May, 2023
Published online: June, 2023

KEYWORDS
Regression Test Case Prioritization.

INTRODUCTION

As the world is fully embracing the digital era known as the fourth industrial revolution, more and more observers and researchers recommend exploring programming skills; the result of a programming task is the production of a software/Application (App). The software development life cycle (SDLC) serves as a guide to the whole software development process. One of the mandatory steps in SDLC is testing the software before use and ensuring that the intended purpose is achieved even after deployment and possible amendments or updates(*Software Development Life Cycle: The Phases of SDLC - TestLodge Blog*, n.d.).

Software testing is one of the key components of software engineering activities performed during and after software development. Generally, Software maintenance activity is an expensive phase that can amount to 60% of the total cost of software production(Musa, 2014); software testing takes 15% out of the software maintenances percentage.

Akhtar (2021) indicates that the fundamental target of software tests is to find and remove bugs, which improves the performance of the software, user experience, security, etc. It is important to note that software is the driving force of the digital economy that the world is exploring; software applications will be everywhere - Airports, water treatment plants, hospitals, traffic, and be the e-Transformation backbone. Software errors can lead to severe harm, such as loss of lives, security issues, loss of money, and even disaster in some cases; therefore, it is dangerous if software testing is not given great importance. To a software end-user, it is a breach of contract to deliver or launch a software application without properly testing it, as it might lead to immediate small or big problems in the future.

Regression testing is one of the functional testing activities done to ensure that changes resulting from fixing errors or updating the software do not affect the initial functionalities and requirements of the software. It involves using test cases which is the set of conditions under which a tester will determine whether the software is working as expected. The set of conditions includes a set of inputs, execution conditions, and expected

Corresponding author: Hassan Abubakar
hassangummi@gmail.com
Department of Mathematical Sciences, Abubakar Tafawa Balewa University, Bauchi

results. A particular software can have many test cases collectively called test suites. Regression testing almost takes 80% of the overall testing budget and up to 50% of the cost of software maintenance (Musa et al., 2016)

BACKGROUND

The live circle of a software system involves changes that occur through an upgrade to a new version or update for better performance or additional futures; this change has to be retested to ensure stability and quality of the software is not loss

Regression testing is one of the functional testing activities done to ensure that changes resulting from fixing errors or updating the software do not affect the initial functionalities and requirements of the software. It involves using test cases which is the set of conditions under which a tester will determine whether the software is working as expected. The set of conditions here includes a set of inputs, execution conditions, and expected results. A particular software can have many test cases collectively called test suites. Regression testing almost takes 80% of the overall testing budget and up to 50% of the cost of software maintenance(Musa et al., 2016). There are three main branches of regression testing techniques (Yoo et al., 2010) these are: Regression test case selection, Regression test suite minimization, and Regression test case prioritization

Test Case Selection

Test Case Selection also targets eliminating redundant or unnecessary test data selecting parts of tests from the test suite that are most relevant. (Jin et al., 2010) indicates that the easiest approach for the tester is to execute all the current tests to be assured that all new changes are harmless; this technique is referred to as the retest-all technique. But this is only achieved if the test suite is small in size. The bottleneck of the Test Case Selection Technique is that if the selection is made randomly, it can bring about checking little portions of the altered programming. The selected Test Case might not be related to the altered program. Regression tests selection techniques for object-oriented programs are categorized into three categories by namely: Firewall-based regression tests selection, Program model-based regression tests selection, and Design-based regression tests selection (Musa et al., 2016)

Test Case Minimization

Test suite minimization is the technique that involves removing redundant test cases to reduce test suites long-running. Test cases are assumed to be redundant or obsolete as the result of their input/output relationship is no more meaningful to the changes in the program. So also, test cases become obsolete when new test cases are developed based on the updated version of the program or when the test cases structure no longer conforms to the requirements of the program under test (Yoo et al., 2010). The only disadvantage of the minimization techniques is that they might reduce the faults detection capability of that test suite(Bello et al., 2019).

Test Case Prioritization

Test Case Prioritization enhances the fault detection rate by meeting two important factors: time and budget in software testing. A TCP involves scheduling test cases to detect errors with the highest severity first than those with low severity (Huang et al., 2015). TCP enables effective utilization of testing resources when the resources are limited or insufficient to retest-all the test suite. A prioritization technique aid in finding out more bugs under a limited resource-constrained situation and, in return, enhance the system's reliability quickly(Kumar et al., 2015). Eliminating bugs and avoiding high-risk defects as early as possible are the main advantages of TCP (Musa et al., 2016)

SOME EVALUATION MEASURES FOR TEST CASE PRIORITIZATION

APFD

APFD is the most widely used performance metric used in TCP. Even though it is still in use, researchers have highlighted many weaknesses. (Paterson, 2019) shows that APFD is not a perfect assessment of how effective a test case prioritization technique is. APFD assumes all possible faults in a system under test have been found and of the same cost and severity. In situations where not all faults are found, the impact of finding faults is

Corresponding author: Hassan Abubakar
hassangummi@gmail.com
Department of Mathematical Sciences, Abubakar Tafawa Balewa University, Bauchi

lessened since the `pool' of possible APFD scores is lessened (Paterson, 2019).

NAPFD

NAPFD or Normalized APFD is an upgraded version of APFD that resolves some of the shortfalls of APFD. (Qu et al., 2007) presented NAPFD with modification on APFD by calculating a value p, which stands for the ratio of total faults detected by the whole test suite (Paterson, 2019).

APFDc

APFDc is also an extension of APFD; Cost-effectiveness is made the focus in prioritizing test cases. The main advantage of this metric is the execution cost that is considered alongside fault detection (Bello, 2019).

AN ANALYSIS OF REGRESSION TEST CASE PRIORITIZATION TECHNIQUES

The need to make regression testing more effective in discovering faults and reducing the overall cost is rising as the 4th Industrial Revolution continues. Many researchers are working in this regard. An in-depth analysis of the literature on this topic was carried out to have an inside on travelogues regression testing development. Despite the challenges highlighted by (Fyfe, 2016) in the process of publications in the early 20th century, some publications can still be found regarding the early stages of the general software testing process(DeMillo et al., 1978; Goodenough et al., 1975; Shapiro et al., 1968; Tucker, 1965). Some of the early publications on Regression testing include (Leung, K. N. H., & White, 1988; Leung et al., 1989). Research interest in test case prioritization started and has been steadily rising since the late 90s (Kaushik et al., 2011).

RESEARCH METHODOLOGY

This paper is a comprehensive evaluation of the available literature on the regression testing techniques, intending to synthesize existing knowledge in this field. A literature search similar to (Mukherjee et al., 2021; Rahmani et al., 2021) was conducted to ascertain the latest research trend in regression test case prioritization. A wide range of 20 years was used from 2002 to 2022; 490 papers were analyzed. These are journals from personal archives and those fetched from the analytic academic citation software (*Publish or Perish*, n.d.).

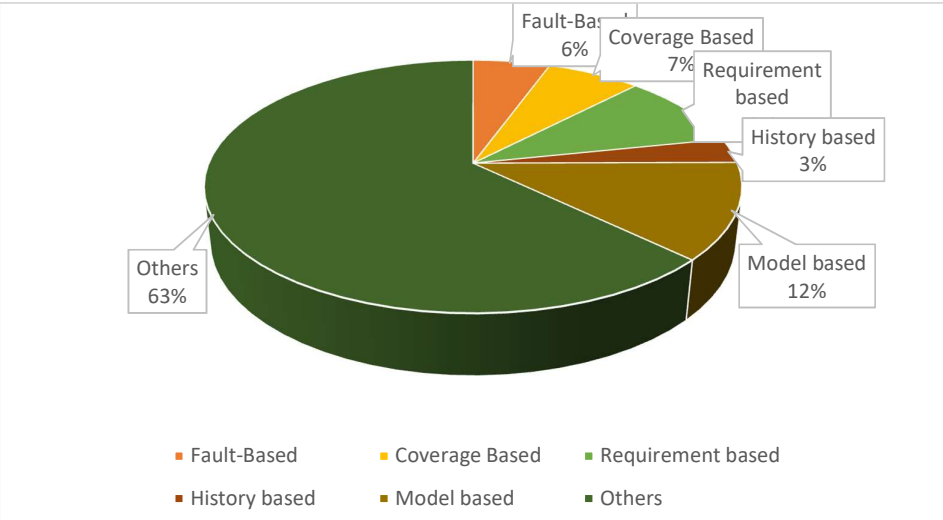

Fig. 1. Distribution of Some Common Prioritization Techniques

The above distribution coincides with findings of (Mukherjee et al., 2021; Rahmani et al., 2021), indicating that Model-based, Coverage based, and Requirements based are the dominant regression test case prioritization by having a higher percentage of researchers

Corresponding author: Hassan Abubakar
hassangummi@gmail.com
Department of Mathematical Sciences, Abubakar Tafawa Balewa University, Bauchi

working using those techniques. It further illustrates that researchers are drastically searching for other techniques different from the common techniques; the portion of other techniques indicates the percentage of the researchers working on different techniques totaling 63%. It includes researchers combining any two techniques to form a more effective technique. It indicates regression test case prioritization takes a new dimension to achieve maximum faults discovery with minimum cost. The dominant techniques are now discussed.

OVERVIEW OF SOME COMMON APPROACHES FOR TEST CASE PRIORITIZATION

Fault Aware Test Case Prioritization techniques

Rothermel et al., 1999) developed fault-based prioritization algorithms for the first time in 1999. The ability of a fault to be exposed by a test case, according to it, not only depends not just on whether a test case executes a certain statement but also on the likelihood that the fault in the test case is found for that test case, the statement will result in failure. Two methods (total fault exposing potential (FEP) and Additional-FEP Prioritization in relation to the fault revealing the test case's potential have been presented in the research. The study also presented four coverage-based approaches. Additional - FEP outscored all other coverage-based techniques. Total FEP surpassed the others prioritization techniques except for branch coverage prioritization. Efficacy and APFD metric used indicates a better fault detection rate, and the results prove this irrespective of the cost of the techniques, either expensive or least expensive.

Fault Aware Test Case Prioritization (FATCP) was proposed by (Kim et al., 2010), which used historical fault information and a fault localization technique. Fault Localization Technique is a debugging technique that identifies the location of a fault or a group of faults in a software. For the most part, FATCP outperformed branch and statement coverage prioritizing approaches in the Siemens program

Coverage Aware Test Case Prioritization Techniques

Coverage-aware prioritizing strategies try to optimize coverage of program pieces (statements, branches, procedures, and so on) by a test case and require detailed source code knowledge. It mostly employs a white box/structural testing strategy. Maximum coverage can be considered an intermediate goal, with an improved fault detection rate as the ultimate goal (Mukherjee et al., 2021). Rothermel et al., (2001) demonstrated that better coverage leads to a higher rate of fault discovery using 9 TCP on 8 C programs. The most interesting finding of this study was that the total coverage techniques outperformed the additional branch/statement coverage techniques for the first seven programs (known as the Siemens Program). Still, the additional branch/statement coverage techniques outperformed the total techniques for the eighth program (space), which is a real and massive program from the European Space Agency. The study's flaw was that it deemed all flaws to be of equal severity and did not account for cost.

Another experiment was carried out by(Fang et al., 2012), focusing on logic coverage testing. Logic coverage testing methods include branch coverage and modified condition/decision coverage. For safety-critical software, logic coverage is universally acknowledged. The authors coined the term CI (Convergence Index), which is used to determine when to terminate testing. A quick convergence indicates a low cost of testing and a high rate of defect identification.

More recently (Mahdieh et al., 2020) set a goal of their research to improve coverage-based TCP approaches by considering the fault-proneness distribution across code units. They use the software's bug history to introduce a defect prediction method to learn a neural network model. The methodology offered is a broad concept that may be used to various coverage-based approaches, including the additional and total TCP methods. The evaluation of the empirical study shows that applying the proposed approach enhances the additional strategy greatly.

Coverage aware TCP techniques continue to attract researcher's attention towards improving the effectiveness of TCP in general(Rahmani et al., 2021). Although the general assumption of coverage-aware TCPs seems to be fair that the higher coverage

Corresponding author: Hassan Abubakar
hassangummi@gmail.com
Department of Mathematical Sciences, Abubakar Tafawa Balewa University, Bauchi

equals greater efficacy, coverage is only one factor to consider when prioritizing test cases. A high level of coverage does not always imply that all flaws are covered. TCP approaches should be time, cost, and requirement oriented, as the major purpose of testing is to ensure the delivery of a quality product within a certain time and budget.

Requirement and Risk-Oriented Test Case Prioritization Techniques

Starbase Corporation (Mogyorodi, 2001) discussed the Requirement Based Testing (RBT) approach with CaliberRBT, which can build a small number of test cases from requirements. The author provided an issue with 137,438,953,472 test cases, and CaliberRBT solved the problem with only 22 test cases. When an issue is discovered during the requirements phase, the cost of fixing it is the lowest. According to the bug report distribution, the requirements phase accounts for 56 per cent of all defects, while the design and coding phases account for 27 per cent and 7%, respectively (Mogyorodi, 2001). Testing can be done in parallel with development with the help of RBT. Testing is not a bottleneck in this scenario. According to (Uusitalo et al., 2008), testing and requirements engineering are inextricably intertwined

Arafeen et al., (2013) investigated whether test cases may be clustered based on requirement similarity. The requirement clusters are prioritized, and test cases from higher priority clusters are chosen in greater numbers. Prioritization of test cases based on requirements was reported by (Srikanth et al., 2016). Test cases are mapped to the software requirements they test and then prioritized based on the mapped requirements' attributes, such as customer-assigned priority and implementation complexity. The fact that requisite attributes are often estimated and subjective values is one potential shortcoming of this technique.

Cost Cognizant Test Case Prioritization Techniques

We discussed the Savings Factor while exploring coverage-aware prioritization prototypes (SF). Calculating SF is essentially a conversion of the APFD into a benefit scale. Test case prioritization can save money, but it comes with execution/implementation costs. Building a cost model is now required, and we'd want to focus on five articles that evaluate cost-benefit tradeoffs for TCP approaches in this paragraph.

The fundamental APFD measure has two limitations in terms of cost-awareness. To begin with, it views all flaws to be equally serious. Second, it assumes that each test case costs the same amount of money in terms of resources. (Elbaum et al., 2001) modified the original APFD metric to APFDc, which considers the fault detection rate, the severity of discovered defects, and the cost of running test cases.

Model Based Test Case Prioritization Techniques

In terms of time, money, and resources gathering execution data is costly. Additionally, as source code evolves to meet new requirements, execution metadata must be updated regularly, making maintenance onerous. In such instances, TCP approaches based on models gain traction. Model-based TCP is a grey box-oriented technique in which specification models (state diagrams, activity diagrams, and so on) are used to depict intended system behavior. Each test case is associated with a model execution path. The term grey-box is used when knowledge about the system's underlying data structure and architecture is required but not the source code. Model-based TCP is a good choice since model execution is faster than system execution. Model-based testing (MBT) is gaining popularity as a test automation technique. We'd like to highlight five notable publications focused on model-based test case selection in this area.

A model-based prioritization technique was presented by (Fraser et al., 2007). Their prioritization method is based on the idea of property relevance (Fraser et al., 2006). If it is theoretically plausible for a test case to violate a model property, the test case is relevant. A model-checker is utilized as the input to the greedy algorithm to obtain the relevance relation. While they demonstrated that property-based prioritizing outperforms coverage-based prioritization, they also stated that the effectiveness of property-based

Corresponding author: Hassan Abubakar
hassangummi@gmail.com
Department of Mathematical Sciences, Abubakar Tafawa Balewa University, Bauchi

prioritization is strongly reliant on the model specification's quality.

Other test case prioritization approaches

We want to highlight other distinct prioritization approaches, as we have documented our representative articles from each avenue of test case prioritization techniques in the previous subsections. Prioritizations based on search algorithms, ant colony optimization, clustering-oriented prioritization, multi-objective prioritization, and prioritizations without coverage information are all gaining popularity. For TCP, (Li et al., 2007) used metaheuristics and evolutionary algorithms (Li et al., 2007). For test case prioritizing, Li et al. conducted simulated tests on Search Algorithms. The researchers looked at five different search algorithms (Total Greedy, Additional Greedy, 2- Optimal Greedy, Hill Climbing, and Genetic Algorithms). Prioritizing test cases can also be done using the ACO (Ant Colony Optimization) technique(Agrawal et al., 2018; Singh et al., 2010).

Hao et al., (2014) proposed a unified strategy in 2014 that combined total and additional (yet to be covered) techniques. (Leon et al., 2003) developed new prioritization techniques that combine coverage-based and distribution-based approaches. This hybrid approach is based on the observation that basic coverage maximization outperforms repeated coverage maximization fairly well. (Elbaum et al., 2000) proposed a prioritization technique in which, after achieving 100 per cent coverage, test cases are re-prioritized starting from zero per cent coverage. On the other hand, basic coverage maximization ceases prioritizing once 100 per cent coverage is reached. According to (Leon et al., 2003), the fault-detection rate of repeated coverage maximization is lower than basic coverage maximization. It prompted them to explore a hybrid approach in which test cases are prioritized based on coverage first, then switched to distribution-based prioritization once the basic coverage maximization is complete. They looked at two alternative strategies based on distribution. The one-per-cluster method selects one test case from each cluster and ranks them according to the clusters' order formed during clustering. The failure-pursuit method works similarly, but it also includes the k nearest neighbors of every test case that finds a flaw. The findings revealed that distribution-based prioritization techniques outperformed repeated coverage maximization techniques.

In line with the trend of combining techniques for a more effective one that dominated the statistics in the Others category, some of the research work is hereby reviewed. Id et al., (2021) indicates the recent confidence developed by researchers on Machine Learning (ML) techniques towards improving the effectiveness of Test case Selection and Prioritization (ML-based TSP). A relatively accurate prediction model is achieved using test cases information from both partial and imperfect sources with the help of this technique. Ranking models, NLP-based, RL, and clustering are the most commonly used ML techniques in TSP. Combining two of these techniques, for example, Ranking models and NLP-based were reported severally by researchers. The combination improves test

Corresponding author: Hassan Abubakar
hassangummi@gmail.com
Department of Mathematical Sciences, Abubakar Tafawa Balewa University, Bauchi

case prioritization performance, even though the study reports a lack of standard evaluation procedures and appropriate publicly hindered ML-based TSP performance assessment, directly affecting the knowledge of the current open problems and state-of-the-art techniques. Researchers such as (Ramírez et al., 2022) put effort into addressing this issue. The researcher's work also indicates the good chances of an increase in test case prioritization effectiveness through the combination of the available techniques

In another study (Samad et al., 2021), the researchers proposed a multi-objective approach using swarm optimization (MOPSO). The MOPSO algorithm uses code Coverage, execution time, and fault detection to prioritize test cases. The researchers report an outstanding performance in terms of less cost, extreme fault detection, and highest coverage in comparisons with reverse ordering, random ordering, and no ordering.

Mishra et al., (2019), the researchers implement a multi-objective genetic algorithm in test case optimization and prioritization. Statement coverage, requirement priority, execution time, and risk exposures were the factors considered to be associated with test cases for a specific SUT at a stipulated time constraint. The proposed technique uses a multi-criteria based GA to prioritize and optimize test cases by considering a stipulated time to execute test cases.

Bello et al. (2019) proposed a multi-criteria evolutionary regression test Prioritization for dynamic object-oriented programs. The technique implements a genetic algorithm with a single objective multi-criteria fitness function to find the best possible ordering of the test suite to a very round of repetition and then produce the prioritized test suite as the solution. Considering more than one parameter (criterion) is an additional future of this technique. These parameters are then assigned weight and later summed up to compute the overall weight considered as the fitness function value of every test case in the test suite. The fitness function values are then used to prioritize the Test Cases. Although the technique has not been implemented, talkless of it has been evaluated (Bello et al., 2019). This research indicates the greatest challenge of handling the dynamic features of Python in testing software systems and the lack of mature tools to handle it in all the development phases. Furthermore, the research direction is not cost-cognizant because the performance matrix they plan to use is APFD, which considers the cost and severity of a fault to be uniform.

The severities of dependent fault are used to assign a value to a test case while other information about the test cases is retrieved from a repository. This approach will pass the complete information about the test cases as input into the ECRTP algorithm. The result will be a scheduled test case order that can detect severe faults earlier during regression testing, and the best-ordered existing test suite will be saved for further reuse in a test case repository. A holistic analysis was conducted to compare this approach with three existing approaches using 8 Java programs to ascertain the most effective cost-cognizant approach. Test effort efficiency, fault detection effectiveness, and APFDc measurements were used as the basis of the experimental data, statistical analysis, evaluation, and comparisons of the proposed approach. All the results clearly show that the ECRTP approach outperforms the other approaches, hence proving the worthiness of ECRTP in Regression testing. However, the researcher indicates the need to extend this work using the latest regression testing information. Also, to consider other object-oriented metrics such as Coupling and Cohesion for better evaluation and incorporate multi-objective evolutionary processes to increase the effectiveness of this technique. Also, due to rapid growth in the number of developers in Dynamic OOP such as Phyton, extending this technique for dynamic object-oriented languages such as Python is highly needed; this forms the basis of my choice to extend this work to address the limitations mentioned above.

Wang et al., (2017) argued that due to the widespread practice in existing coverage-based TCP techniques that focuses on maximum codes coverage rather than the distribution of faults in the source code, which is the actual case, this could lead to fewer faults detection. They further emphasized that test cases from faulty source code should have higher priorities because they are more likely to find faults. They proposed an RTP approach that puts more weight on the fault-inclined

Corresponding author: Hassan Abubakar
hassangummi@gmail.com
Department of Mathematical Sciences, Abubakar Tafawa Balewa University, Bauchi

source code based on this assumption. It will ensure test cases that traverse those fault-inclined areas will be assigned higher priority and are first to be ordered during the regression testing process. A statistic defect prediction model and a static bug-finder were deployed to detect fault-inclined source code before adapting the existing coverage-based TCP algorithm. The evaluation results indicate that this technique is almost 10% higher than traditional coverage-based techniques. Although this technique indicates a certain level of improvement in this area of research, there is an element of doubt that it will be biased in identifying fault-inclined source code areas. It is also a Static OOP specific and non-cost-cognizant technique hence the need for further exploration of this area to accommodate Dynamic OOP users

Musa et al., (2016) propose an effective regression test case prioritization approach using GA with a reduction in fault severity so far as the previous test cases execute the statement. The approach considers selected test cases extracted from affected statements and coverage information. The approach prioritizes this set of selected test cases using a genetic algorithm considering a 5% reduction in fault severity reduction from the initial fault severity. APFD metric was used to evaluate the effectiveness of this approach on three sample programs with three different versions of each. Considering Rd = 5% in fault severity, the evaluation results show better results in the nine programs considering the APFD metric used. Using the measured performance using APFD metric gotten from the results, the prioritized selected test case using GA with reduced severity of fault is more effective compared to a prioritize selected test cases using GA with the same severity of fault; with an overall average increase in the APFD metric value by 10%.

(Umbarkar et al., 2015) explore the use of a Genetic Algorithm for automatic test case generation and prioritization. The researchers use a fitness function that considers multi-criteria that measure multiple control flow data. Faults detected by test cases and the Faulty severity and coverage based data value are the factors they consider for their prioritization technique. Their experimental result outperformed other previously chosen works for comparison based on their similarities.

Prakash et al., (2014) developed another test case prioritization that uses a multiple criteria coverage method to improve test efficiency. Requirement priority factor value, total statement coverage, and total risk exposure value were the factors considered for prioritization, while the total execution time was considered for multi-objective constraint in the minimization process. The rate of fault detection capacity for numerous SUT and the general performance of regression testing drastically improved

CONCLUSION

This research paper delves into regression testing issues, test case prioritization precisely. There is numerous basis on which test case prioritize built upon, and they are Model-based, coverage-based and Requirements-based, Mutation-based, fault-based, among others; the first three are dominant in this research domain. The researcher's interest is turning to some other approaches that are not common and combining two techniques to develop a more effective technique. Many researchers have already worked on combining coverage with fault detection ability. Test case prioritization targets cost, time, and effort metrics reduction and maximizes customer satisfaction. The need to strengthen its effectiveness of Test prioritization is of great importance due to the exponential growth in software usage. When looking at most of the research performed in test case prioritization, their main focus is to develop new ideas that will give maximum code coverage and minimum execution cost, leading to a high-quality product. These findings will be concluded with a recommendation to the researcher communities to explore other approaches to enhance test case prioritization effectiveness, including combining multiple techniques. In line with this, we are currently conducting a research title "Quality-aware GA based cost cognizant test case prioritization for object-oriented programs" to contribute to this research domain

Reference

Agrawal, A. P., and Kaur, A. (2018). A comprehensive comparison of ant

Corresponding author: Hassan Abubakar
hassangummi@gmail.com
Department of Mathematical Sciences, Abubakar Tafawa Balewa University, Bauchi

colony and hybrid particle swarm optimization algorithms through test case selection. *Data Eng*, 397–405.

Arafeen, M. J., and Do, H. (2013). Test case prioritization using requirements-based clustering. *Proceedings - IEEE 6th International Conference on Software Testing, Verification and Validation, ICST 2013*, 312–321. https://doi.org/10.1109/ICST.2013.12

Bello, A., Md. Sultan, A. B., and Shehu, S. (2019). Multi-Criteria Evolutionary Regression Test Prioritization for Dynamic Object-Oriented Programs. *International Journal of Advances in Electronics and Computer Science*, *6*(1), 14–18.

DeMillo, R., Lipton, R., Computer, F. S.-, and 1978, U. (1978). Hints on test data selection: Help for the practicing programmer. *Ieeexplore.Ieee.Org*. https://ieeexplore.ieee.org/abstract/document/1646911/

Elbaum, S., Malishevsky, A. G., and Rothermel, G. (2000). Prioritzing test cases for regression testing. *Proceedings of the ACM SIGSOFT 2000 International Symposium on Software Testing and Analysis*, 102–112. https://doi.org/10.1145/347636.348910

Elbaum, S., Malishevsky, A., and Rothermel, G. (2001). Incorporating varying test costs and fault severities into test case prioritization. *Proceedings - International Conference on Software Engineering*, 329–338. https://doi.org/10.1109/ICSE.2001.919106

Fang, C. ., Z., C., and B., X. (2012). *Comparing Logic Coverage Criteria on Test Case*. *0*(0), 0–14.

Fraser, G., and Wotawa, F. (2006). Property relevant software testing with model-checkers. *SIGSOFT Software Engineering Notes*, *31*(6), 1–10.

Fraser, G., and Wotawa, F. (2007). Test-case prioritization with model-checkers. *SE'07: Proceedings of the 25th Conference on IASTED International Multi-Conference*, 267–272.

Fyfe, A. (2016). Journals and Periodicals. *A Companion to the History of Science*, 387–399. https://doi.org/10.1002/9781118620762.CH27

Goodenough, J. B., and Gerhart, S. L. (1975). Toward a Theory of Test Data Selection. *IEEE Transactions on Software Engineering*, *SE-1*(2), 156–173. https://doi.org/10.1109/TSE.1975.6312836

Hao, D., Zhang, L., Zhang, L., Rothermel, G., and Mei, H. (2014). A unified test case prioritization approach. ACM Trans. *Softw. Eng*, *24*(1), 10–31.

Huang, R., Chen, J., Towey, D., Chan, A. T. S., and Lu, Y. (2015). Aggregate-strength interaction test suite prioritization. *Journal of Systems and Software*, *99*, 36–51. https://doi.org/10.1016/j.jss.2014.09.002

Id, D., Pan, R., Bagherzadeh, M., Ghaleb, T. A., and Briand, L. (2021). *Test Case Selection and Prioritization Using Machine Learning: A Systematic Literature Review*. *1*, 1–32.

Jin, W., Orso, A., and Xie, T. (2010). Automated behavioral regression testing. *ICST 2010 - 3rd International Conference on Software Testing, Verification and Validation*, 137–146. https://doi.org/10.1109/ICST.2010.64

Kaushik, N., Salehie, M., Tahvildari, L., Li, S., and Moore, M. (2011). Dynamic prioritization in regression testing. *Proceedings - 4th IEEE International Conference on Software Testing, Verification, and Validation Workshops, ICSTW 2011*, 135–138. https://doi.org/10.1109/ICSTW.2011.68

Kim, S., and Baik, J. (2010). An effective fault aware test case prioritization by incorporating a fault localization technique. *10, p. 1*.

Kumar, H., and Chauhan, N. (2015). A Module Coupling Slice Based Test Case Prioritization Technique. *International Journal of Modern Education and Computer Science*, *7*(7), 8–16.

Leon, D., and Podgurski, A. (2003). A comparison of coverage-based and distribution-based techniques for filtering and prioritizing test cases. *Proceedings - International Symposium on Software Reliability Engineering, ISSRE*, *2003-Janua*, 442–453.

Corresponding author: Hassan Abubakar
hassangummi@gmail.com
Department of Mathematical Sciences, Abubakar Tafawa Balewa University, Bauchi

https://doi.org/10.1109/ISSRE.2003.1251065

Li, Z., Harman, M., and Hierons, R. M. (2007). Search algorithms for regression test case prioritization. *IEEE Transactions on Software Engineering*, *33*(4), 225–237. https://doi.org/10.1109/TSE.2007.38

Mahdieh, M., Mirian-Hosseinabadi, S. H., Etemadi, K., Nosrati, A., and Jalali, S. (2020). Incorporating fault-proneness estimations into coverage-based test case prioritization methods. *Information and Software Technology*, *121*. https://doi.org/10.1016/j.infsof.2020.106269

Mishra, D. B., Mishra, R., Acharya, A. A., and Das, K. N. (2019). Test case optimization and prioritization based on multi-objective genetic algorithm. In *Advances in Intelligent Systems and Computing* (Vol. 741). Springer Singapore. https://doi.org/10.1007/978-981-13-0761-4_36

Mogyorodi, G. (2001). requirements-based testing: an overview caliberrbt/ mercury interactive integration overview. *Integrat. VLSI J.*, 286–295.

Mukherjee, R., and Patnaik, K. S. (2021). A survey on different approaches for software test case prioritization. *Journal of King Saud University - Computer and Information Sciences*, *33*(9), 1041–1054. https://doi.org/10.1016/j.jksuci.2018.09.005

Musa, S. (2014). A Regression Test Case Selection and Prioritization for Object-Oriented Programs using Dependency Graph and Genetic Algorithm. *International Journal of Engineering And Science*, *4*(7), 54–64.

Musa, S., Sultan, A. B. M. D., Abdul Ghani, A. A. Bin, and Baharom, S. (2016). Regression Test Cases Prioritization for Object-Oriented Programs Using Genetic Algorithm with Reduced Value of Fault Severity. *International Journal of Soft Computing*, *11*(4), 247–254.

Prakash, N., and Gomathi, K. (2014). Improving Test Efficiency through Multiple Criteria Coverage Based Test Case Prioritization. *International Journal of Scientific & Engineering Research*, *5*(4), 420–424.

Publish or Perish. (n.d.). Retrieved January 25, 2022, from https://harzing.com/resources/publish-or-perish

Rahmani, A., Ahmad, S., Jalil, I. E. A., and Herawan, A. P. (2021). A Systematic Literature Review on Regression Test Case Prioritization. *International Journal of Advanced Computer Science and Applications*, *12*(9), 253–267. https://doi.org/10.14569/IJACSA.2021.0120929

Ramírez, A., Informática, D. De, and Insti-, U. D. C. (2022). ATaxonomy of Information Attributes for Test Case Prioritisation: Applicability,Machine Learning. *ACM Transactions on Software Engineeringand Methodology*, *1*(1). https://doi.org/10.1145/3511805

Rothermel, G., Untch, R. H., Chu, C., and Harrold, M. J. (1999). Test Case Prioritization: an Empirical Study. *Proceedings of the IEEE International Conference on Software Maintenance*, 179. https://doi.org/10.1109/ICSM.1999.792604

Rothermel, G., Untch, R. H., Chu, C., Harrold, M. J., and Society, I. C. (2001). Prioritizing Test Cases For Regression Testing Prioritizing Test Cases For Regression Testing. *IEEE Transactions on Software Engineering*, *27*(10), 929–948. https://doi.org/10.1145/347324.348910

Samad, A., Mahdin, H. Bin, Kazmi, R., Ibrahim, R., and Baharum, Z. (2021). Multiobjective Test Case Prioritization Using Test Case Effectiveness: Multicriteria Scoring Method. *Scientific Programming*, *2021*. https://doi.org/10.1155/2021/9988987

Shapiro, S. S., Wilk, M. B., and Chen, H. J. (1968). A Comparative Study of Various Tests for Normality. *Journal of the American Statistical Association*, *63*(324), 1343–1372. https://doi.org/10.1080/01621459.1968.10480932

Singh, V. B., Kapur, P. K., and Tandon, A. (2010). Measuring reliability growth of software by considering fault

Corresponding author: Hassan Abubakar
✉ hassangummi@gmail.com
Department of Mathematical Sciences, Abubakar Tafawa Balewa University, Bauchi

dependency, debugging time Lag functions and irregular fluctuation. *SIGSOFT Softw. Eng. Notes*, *35*(3), 1–11. https://doi.org/http://dx.doi.org/http://dx.doi.org/10.1145/1764810.1764831

Software Development Life Cycle: The phases of SDLC - TestLodge Blog. (n.d.). Retrieved September 23, 2021, from https://blog.testlodge.com/software-development-life-cycle/

Srikanth, H., Hettiarachchi, C., and Do, H. (2016). Requirements based test prioritization using risk factors: An industrial study. *Information and Software Technology*, *69*, 71–83. https://doi.org/10.1016/j.infsof.2015.09.002

Tucker, A. E. (1965). *The Correlation of Computer Program quality with testing effort*. https://scholar.google.com/scholar?hl=en&as_sdt=0%2C5&q=A.+E.+Tucker%2C+%22The+Correlation+of+Computer+Program+Quality+with+Testing+Effort%2C%22+System+Development+Corporation%2C+TM+2219%2F000%2F00%2C+January+1965.&btnG=

Umbarkar, A. J., and Sheth, P. D. (2015). Crossover Operators in Genetic Algorithms: a Review. *ICTACT Journal on Soft Computing*, *06*(01), 1083–1092. https://doi.org/10.21917/ijsc.2015.0150

Uusitalo, E. J., Komssi, M., Kauppinen, M., and Davis, A. M. (2008). Linking requirements and testing in practice. *Proceedings of the 16th IEEE International Requirements Engineering Conference*, 265–270.

Yoo, S., and Harman, M. (2010). Regression testingminimization, selection and prioritization: asurvey. *Software Testing Verification and Reliability*, *22*, 67–120. https://doi.org/10.1002/stvr

Corresponding author: Hassan Abubakar
✉ hassangummi@gmail.com
Department of Mathematical Sciences, Abubakar Tafawa Balewa University, Bauchi