2024, 21(2 Special Issue): 0802-0811
https://doi.org/10.21123/bsj.2024.9710
P-ISSN: 2078-8665 - E-ISSN: 2411-7986

Baghdad Science Journal

Hybrid Framework To Exclude Similar and Faulty Test Cases In Regression Testing

Muhammad Asim Siddique*, Wan M.N. Wan-Kadir, Johanna Ahmad, Noraini Ibrahim

Faculty of Computing, Universiti of Teknologi, Johor Bahru, Malaysia.
*Corresponding Author.

PARS2023: Postgraduate Annual Research Seminars 2023.

Received 27/09/2023, Revised 10/02/2024, Accepted 12/02/2024, Published 25/02/2024

Abstract

Regression testing is a crucial phase in the software development lifecycle that makes sure that new changes/updates in the software system don't introduce defects or don't affect adversely the existing functionalities. However, as the software systems grow in complexity, the number of test cases in regression suite can become large which results into more testing time and resource consumption. In addition, the presence of redundant and faulty test cases may affect the efficiency of the regression testing process. Therefore, this paper presents a new Hybrid Framework to Exclude Similar & Faulty Test Cases in Regression Testing (ETCPM) that utilizes automated code analysis techniques and historical test execution data to identity and exclude redundant, similar and faulty test cases from the given regression suite. Our experimental results clearly show the benefits of the ETCPM framework in terms of reduction in the testing time, optimization of the resource allocation, and improvement in the overall quality of regression test suite. ETCPM enables software development teams to achieve faster and reliable regression testing by intelligent exclusion of similar and fault test cases, which yields in reduction in the software delivery cycles and better end user satisfaction.

Keywords: Regression Testing, Software Maintenance, Software Quality, Test Case Prioritization, Test Case Selection.

Introduction

Software testing is a crucial analytical process that provides stakeholders with information about the caliber of a good or service under review[1]. Software development is a crucial step in the software development lifecycle that guarantees the program satisfies customer requirements, specifications, and quality standards. New research techniques have been developed as technology advances, bringing additional challenges, shortcomings, and barriers. Software systems grow in size and complexity, making quality more elusive and unstable. According to Tilley and Floss[1], complex systems make existing research challenges more intricate and give rise to new ones. Because one of their duties is to improve software quality, developers are aware of the aggravation caused by flaws in software and are committed to finding a solution. The limitations of retest-all, growing code and test suite size of software systems, and altering landscape of testing requirements provide the rationale for effective regression testing.

The effectiveness of regression testing techniques is context-dependent Terragni, Cheung, & Zhang[2], like time constraints, incremental resource availability and time to release for products. The method of selecting test cases was based on using coverage facts capable of identifying faults as a substitute[3]. The test suite size is used as reduction criteria for test case reduction techniques[4]. Thus, time, coverage, redundancy, and fault detection ability play an important role in regression testing as adequacy criteria as well as the selection, prioritization, or ordering criteria with respect to the context chosen for the technique under study[5]. The code change information is an additional parameter for test case selection methods. However, reduction and prioritization techniques ignore code changes during the regression testing process[6].

The time, coverage, redundancy, and fault detection ability are not strongly correlated[7]. Regression Test Case Selection (RTS) methods are categorized into three types based on their effectiveness: single-criterion, bi-criteria, or multi-criteria approaches. The single objective selection techniques fulfill one aspect (time, coverage, fault detection ability, and redundancy) but ignore the other two effectiveness measures. The multi-objective selection techniques focus on two or more effectiveness measures simultaneously. The measurement of the relationship between these effectiveness measures remains an open problem[8].In Fig.1. regression testing process is further explained.

Retest all
Regression Test Selection
Prioritization of Test Cases

Figure 1. Regression testing

Various models, frameworks, and automated solutions have been developed and suggested to implement techniques for multi-objective test case selection, prioritization, and reduction[9]. The basic idea behind these models and frameworks is based on the relationship between effectiveness measures. These effectiveness measures are time and efficiency[10]. The test case selection techniques impose an additional measure: code change information. Primarily, prioritization and reduction techniques are the subset of the test case selection techniques. The test case selection frameworks[11], coverage-based, fault detector and redundant test case selection frameworks[12], dynamic code changes-based test case selection framework[13] and pairwise feature analysis of product line model [14] are found in literature, techniques and accommodate multi-objective problems. They cannot simultaneously accommodate four effectiveness measures and require a graph-based, code-slicing approach or pre-processing code analysis to bind these effectiveness measures[15]. Furthermore, these frameworks and algorithms are context and language feature-dependent, limiting their generalization property.

To cut down on the expenses of regression testing and enhance its effectiveness (in terms of coverage, fault detection, and redundancy), as well as the time efficiency of the chosen test suite, testers can opt to decrease the test suite's size, limit the number of test case executions, shorten the test case execution duration, choose a subset of test cases previously run on the System Under Testing (SUT), or prioritize the test cases more efficiently [13-15]. Information regarding coverage that includes fault detection capabilities serves as a substitute for choosing regression test cases. The primary goal of these techniques for testers is to enhance the ability to detect faults while reducing the duration of the selected test suite [14]. Regression testing has been widely used to ensure that software evolution does not break existing modules in the system [2]. In fact, effectively selecting test cases and detecting updated changes can be difficult, especially in an environment where test suite solutions are becoming more and more complicated and distributed. [11]. Regression testing is crucial but can be significantly expensive in terms of time and efficiency [12].

While TCP with a prioritizing technique is an example of a hybrid methodology with a high rate of fault detection, TCS with a clustering method lowers cost consumption. A combined effort combining the two approaches might benefit the community more. [8-9] were two of the first researchers to use multiple techniques in their case study. By modifying the software, Malhotra et al. were able to boost confidence in its accuracy [6]. In the meantime, Suri and colleagues were able to shorten the execution time and find the errors sooner than previously [3-4]. Therefore, it has been demonstrated that a hybrid technique is likely to produce a better regression outcome.

The Proposed Hybridization Method

The hybridization consists of two TCS, TCS string similarity-based and search-based. The method begins by undergoing the dataset through the TCS process, followed by TCP. However, in both TCS and TCP, different criteria are examined. In TCS, variable similarity and fault detection are measured, while in TCP, variable weight is measured.

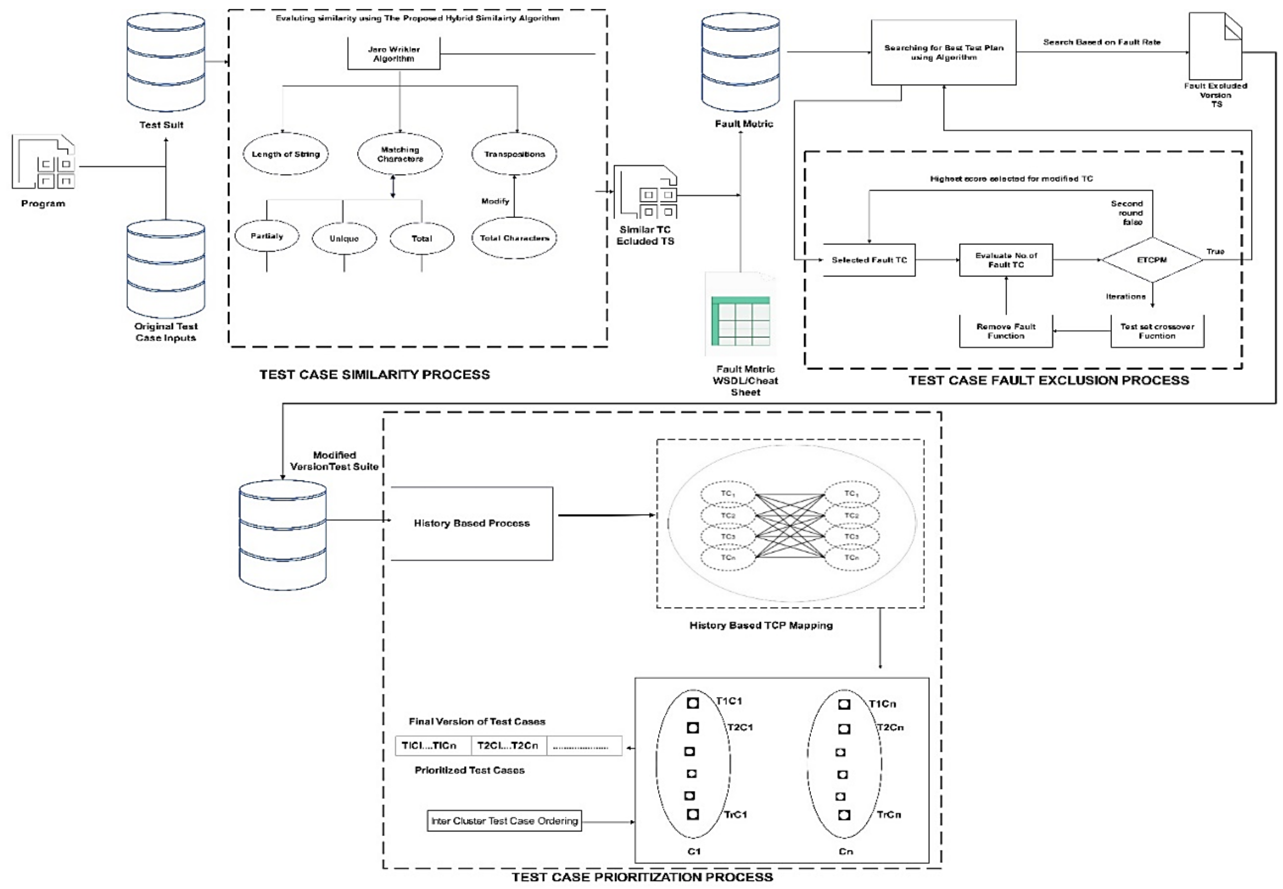
Evaluting similarity using The Proposed Hybrid Similairty Algorithm
Jaro Wrikler Algorithm
Length of String
Matching Characters
Transpositions
Modify
Partialy
Unique
Total
Total Characters
Test Suit
Program
Original Test Case Inputs
TEST CASE SIMILARITY PROCESS
Similar TC Ecluded TS
Fault Metric
Fault Metric WSDL/Cheat Sheet
Searching for Best Test Plan using Algorithm
Search Based on Fault Rate
Fault Excluded Version TS
Highest score selected for modified TC
Second round false
Selected Fault TC
Evaluate No.of Fault TC
ETCPM
True
Iterations
Remove Fault Function
Test set crossover Fucntion
TEST CASE FAULT EXCLUSION PROCESS
Modified VersionTest Suite
History Based Process
TC1
TC2
TC3
TCn
History Based TCP Mapping
Final Version of Test Cases
TICI....TICn
T2CI....T2Cn
Prioritized Test Cases
Inter Cluster Test Case Ordering
T1C1
T2C1
TrC1
T1Cn
T2Cn
TrCn
C1
Cn
TEST CASE PRIORITIZATION PROCESS

Figure 2. Test case priortization process

From these criteriaas mentioned in Fig.2. the author is planning to benefit each variable to form the best test plan for regression. Similarity helps identify the furthest dissimilarity among test inputs, while detection of faults helps in sorting test cases with a high chance of getting faults. After both were executed, the test plan generated from each approach will hybrid and merge by applying a weight-based approach. Test engineers can set weight on which criteria to prioritize. The result can be in three forms: similarity-based selection, search-based selection, and balance scoring selection. Finally, based on the selection, a final test plan is generated and ready to execute regression.

Test Case Similarity Reduction

Regression testing involves retesting a software application after modifications to ensure that new defects haven't been presented or existing functionality remains intact. The goal of test case reduction is to reduce the number of test cases required to adequately cover the functionality of a software application. This can be especially helpful when dealing with extensive test suites to save time and resources. The study has improved the Jaro-Winkler algorithm, a string similarity metric used to compare two strings and determine their similarity. It takes into account both character similarity and character positions in the strings, providing a similarity score ranging from 0 to 1, with 1 indicating a perfect match. String algorithms or textual similarity metrics are techniques for measuring the closeness or dissimilarity of two text sequences. These methods are commonly used in record linkage tasks, where a fast and efficient way to calculate overall similarity between two records is needed for large datasets. In simple meaning, a string algorithm is applied when the goal is to measure the similarities between strings. The following functions are considered in the similarity reduction phase.

Identify Similar Test Cases: Enhanced Jaro-Winkler computes the distance between two strings, so on its own it returns a similarity score. To use this, an appropriate threshold must be defined to determine when two test case names or descriptions within the test suite are similar.

Set a Threshold: An example of this might be to set the threshold so that test cases with a Jar-Jar-Winkler similarity score of 0.8+ similar.

Group Similar Test Cases: Group test cases with similarity scores ≧ a given threshold. These will be sets of test cases that cover similar functionality.

Select Representative Test Cases: For each group, select a representative test case. This could be the most comprehensive test case from the group, the most critical one, etc.

Remove Redundant Test Cases: Remove the non-representative test cases from the groups as they are redundant, covering similar functionality as the representative test case.

Updated Test Suite: The condensed test suite now consists of representative test cases from different groups with similar functionalities, effectively decreasing the total number of test cases while maintaining comprehensive coverage.

Faulty Test Cases Exclusion

In this phase Automated Techniques are employed for removing of faulty test cases using enhanced Search -based algorithm in test case selection. In this phase we use automated techniques for identifying and removing the tests which are more probable to produce wrong or erroneous outcomes. Search-based algorithms are used to explore the space of all potential test cases for the System Under Test (SUT) & to seek out that subset of those tests that are most likely to find defects or faults in the SUT. Here is the general form on working of Phase III:

Similarity Removed Modified Test Suit: A Modified Test Suit from previous phase will be imported in this phase for fault reduction.

Execution and Analysis: In generated time monitor, execute the test cases on the system, capturing outputs in the means of a log and any faults or exceptions that occur.

Fault Detection Criteria: Determine criteria for identifying whether or not the test case execution has identified a fault such as a bug, crash, violation of expected behavior, or any other relevant deviation from normal operation. Using different methodologies (equation-based analysis, heuristic rule-based).

Fitness Function: Create a "fitness function" that captures the likelihood of a test case exposing a fault based on factors such as system behavior, code coverage, execution paths exercised and critical paths exercised.

Search Algorithm: adopting a search-based algorithm that iteratively picks and refines the set of test cases. Genetic algorithms, simulated annealing, particle swarm optimization, and other metaheuristic techniques are often applied with this aim. The algorithm should work to maximize the fitness function while minimizing the number of test cases. This effort can require a significant amount of computational horsepower (and possibly cloud computing) when dealing with large, real-world systems.

Faulty Test Cases Elimination: As far as how the algorithm works, once the process begins, it selects which test cases to include in a given round based on their probability to identify faults not bothering with those that wouldn't make a significant contribution to fault detection in review.

Iterative Refinement: Proceeding from there, the algorithm iterates, further refining the set of test cases by considering the feedback received from their execution on the system. As the process continues, the algorithm further learns how to carve away at test cases that are likely to be at fault.

Stopping Criteria: The search algorithm should have some stopping criteria. This could be a maximum number of iterations, a convergence threshold, or other factors indicating that the algorithm has arrived at a satisfactory solution.

Validation and Review: Once the search algorithm is complete, the selected test cases will still require validation. This can be done through manual assessment or other automated validation techniques to ensure that the chosen test cases are a good indication of fault detection effectiveness.

Integration with Testing Process: Finally, integrate the selected test cases into the overall testing process. The fast tests should become a part of an automated testing suite if one exists, but it might also be valuable to keep the tests outside of the automated suite as an additional check.

Proposed Test Case Prioritization using History Based Approach

In this phase, we perform Test case prioritization using a history-based approach. A modified test suit with similarity and faulty test case removed will be used. Test case prioritization using a history-based approach phase utilizes information from prior test executions to determine the sequence in which they should be executed in upcoming testing cycles. This strategy focuses on test cases that have historically been more likely to find faults or failures to enhance testing efficiency. The history-based test case prioritization process works as follow;

Test Execution and Recording: First, the system under test must be exercised by the test suite and during the execution of each test, all pertinent data like execution times, test case outcomes (pass/fail) and possibly any other data that may give greater insight into the behavior of the system is recorded.

Data Collection and Analysis: Next, you have to take the historical data collected during multiple executions; such as execution times, failure rates, and possibly other metrics like code coverage or specific types of defects uncovered.

Feature Extraction: This comes in two pieces: First, you need to extract features from the historical data that will allow you to tell different teset cases apart. For example, you could figure out what the average execution time, failure rate, etc is for every test case, or how often one type of defect was found.

Prioritization Strategy: Then, choose a strategy to prioritize the tests, based on historical data! There are several you can use, based on the project's goals and priorities, such as:

Failure-based: Report on test case runs, analyze how many times each test case has historically resulted in failed unit tests. This approach assumes that if a test case has failed before i.e., on multiple runs it's more likely to reveal issues in the future.

Execution time-based: Prioritize test cases with the fastest historical execution times. This aims to complete testing quicker by running quicker test cases first.

Combination: Combine multiple factors, such as failure rate and execution time, into a weighted formula to generate an overall prioritization score for each test case.

Normalization: Normalize the extracted features to ensure that they are on the same scale. This step is important when using weighted combinations of features.

Scoring and Ranking: Combine multiple factors, such as bugs' past-failure rate and the amount of time it takes to execute the tests, into a weighted formula to generate an overall prioritization score for each test case.

Sorting and Execution: Normalize the features that you just extracted to ensure that they are all on the same scale. This step is important because you're going to use a weighted combination of the features and you want to be comparing apples to apples.

Iterative Improvement: As new test execution data becomes available, update the historical records and adjust the prioritization strategy accordingly. This iterative process helps refine the test case prioritization over time.

Feedback Loop: Continue to assess the history-based approach's efficacy. Keep an eye on whether the strategy results in quicker defect detection, shorter testing durations, or other process enhancements.

Adaptation: Be open to adjusting the prioritization strategy based on changing project requirements, software updates, and evolving testing goals.

History-based test case prioritization leverages: The insight gained from previous testing experiences to make more informed decisions about the order in which test cases are executed. By focusing on test cases that have historically been more effective at finding issues

Proposed Method overview

Initially, the experiment setup did not involve many test cases, but after the experiment was hybridized, the experiment required two test plans (TP) for merging and prioritizing. This process was initially executed once per each test case input, as shown in Fig.2. Then, a general Hybrid process was proposed to execute all subsequent releases of test cases created by TCP on similarity-based and TCP on fault based, as illustrated in Fig.3.

In the initial run, test cases (TCs) are extracted from a vast, indexed database of textual TCs. The input for the TC repository is obtained from the experimental dataset, which, in this case, was sourced from a software infrastructure repository (SIR). Each TC has a single input string that is organized to run the system.
The dataset for the secondary experiment includes both test inputs and a fault matrix. In this experiment, these outputs are utilized in their entirety. The test inputs undergo a similarity process, while the fault matrix is subjected to a fault analysis. The generated test plan is subsequently consolidated and organized based on a combination of relevance criteria, specifically weighted scoring. The weighted scoring is measured according to the scale set by the test engineer. The process begins with prioritization and ends with selection as shown in Fig.3.

- Test Plan creation based on similarity with prioritization technique
- Creating a test plan based on a prioritizing technique error
- Utilize a weight-based strategy to hybridize and apply a selection of the Test Plans produced by the two prior processes.

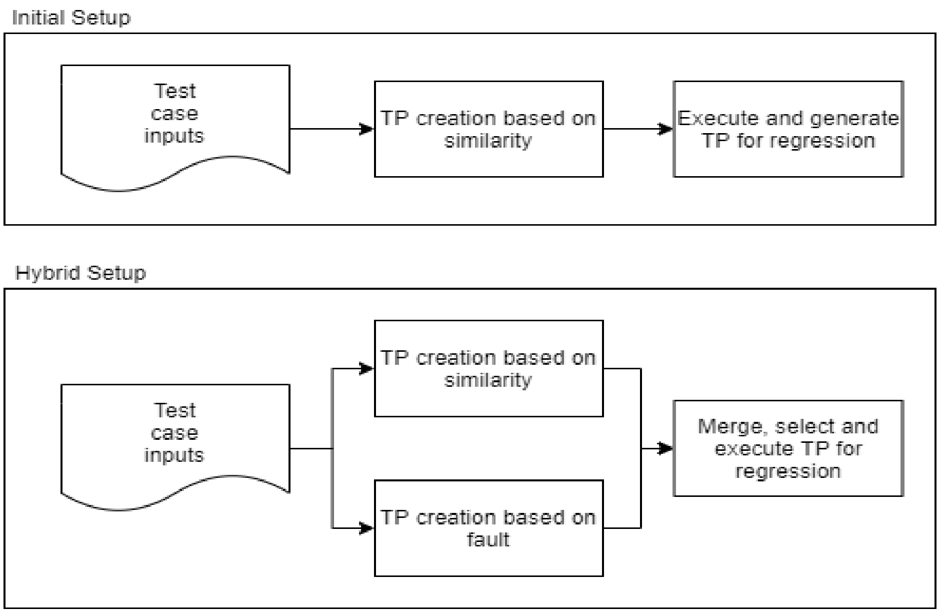
Initial Setup
Test case inputs
TP creation based on similarity
Execute and generate TP for regression
Hybrid Setup
Test case inputs
TP creation based on similarity
TP creation based on fault
Merge, select and execute TP for regression

Figure 3. Initialized hybrid setup

Results and Discussion

The process of the experiment has been elaborated on previously. This study will analyze and compare its result with the existing approach, including a single approach technique.

The applicability of the proposed solution in the experiment is presented. The result of this experiment is described in detail in the sub-section. We have conducted tests on two test suits with 5,10, 15, and 20 test cases in each TS, as shown in Fig.4. and Fig.5.

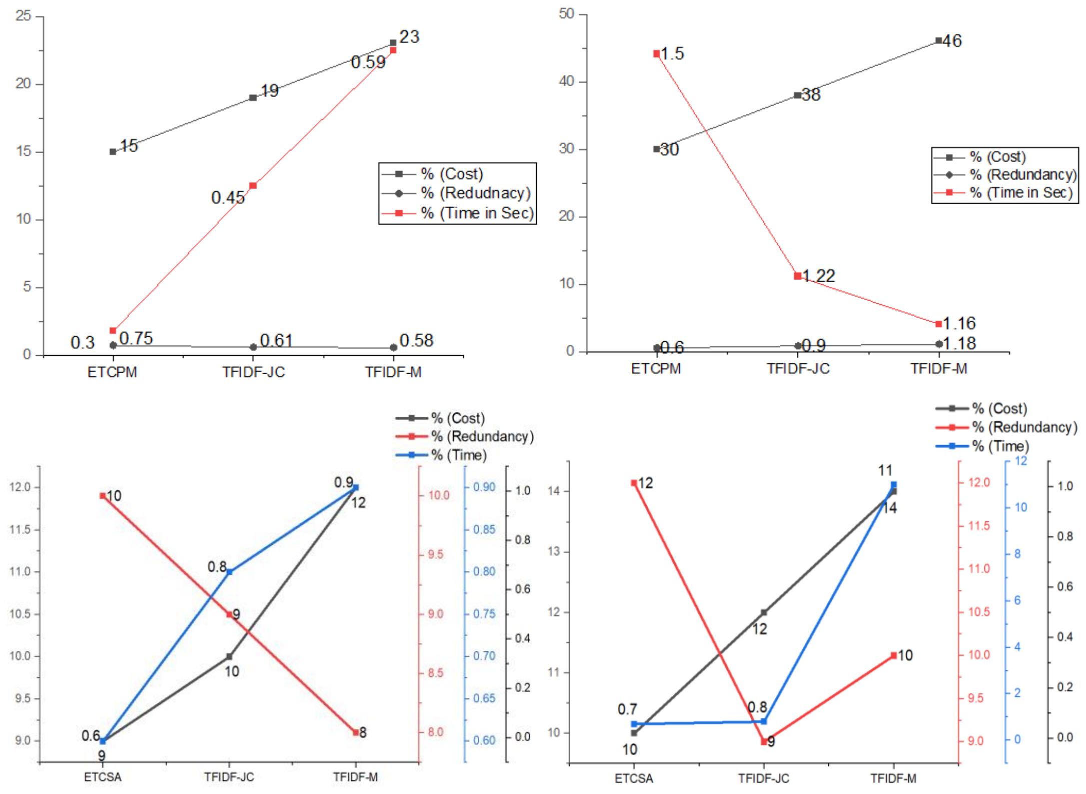

Figure 4. Redundancy, cost & time ratio when test cases = (5, 10)

It is possible to skip certain studies due to the presence of unclear aspects in the field of regression test case selection and test case prioritization. These gray areas pertain to topics like the impact of solution techniques and the interplay between test case selection and prioritization, as well as the connections between testing techniques and testing levels. This study has ties to various communities within the software testing domain, including quality assurance professionals, information systems experts, those involved in service-oriented architecture (SOA), and individuals engaged in testing applications based on object-oriented principles.

An improved method for regression testing to choose and order the test suite To increase fault detection ability and coverage, persistent uncertainty in quality parameter selection and prioritization is necessary to identify relevant and sensitive quality factors like cost, time, and redundancy, which put the business of the software industry at stake. Therefore, the Fault & coverage ratio results when test cases are 5, 10, 15, and 20 are shown in Fig.5.

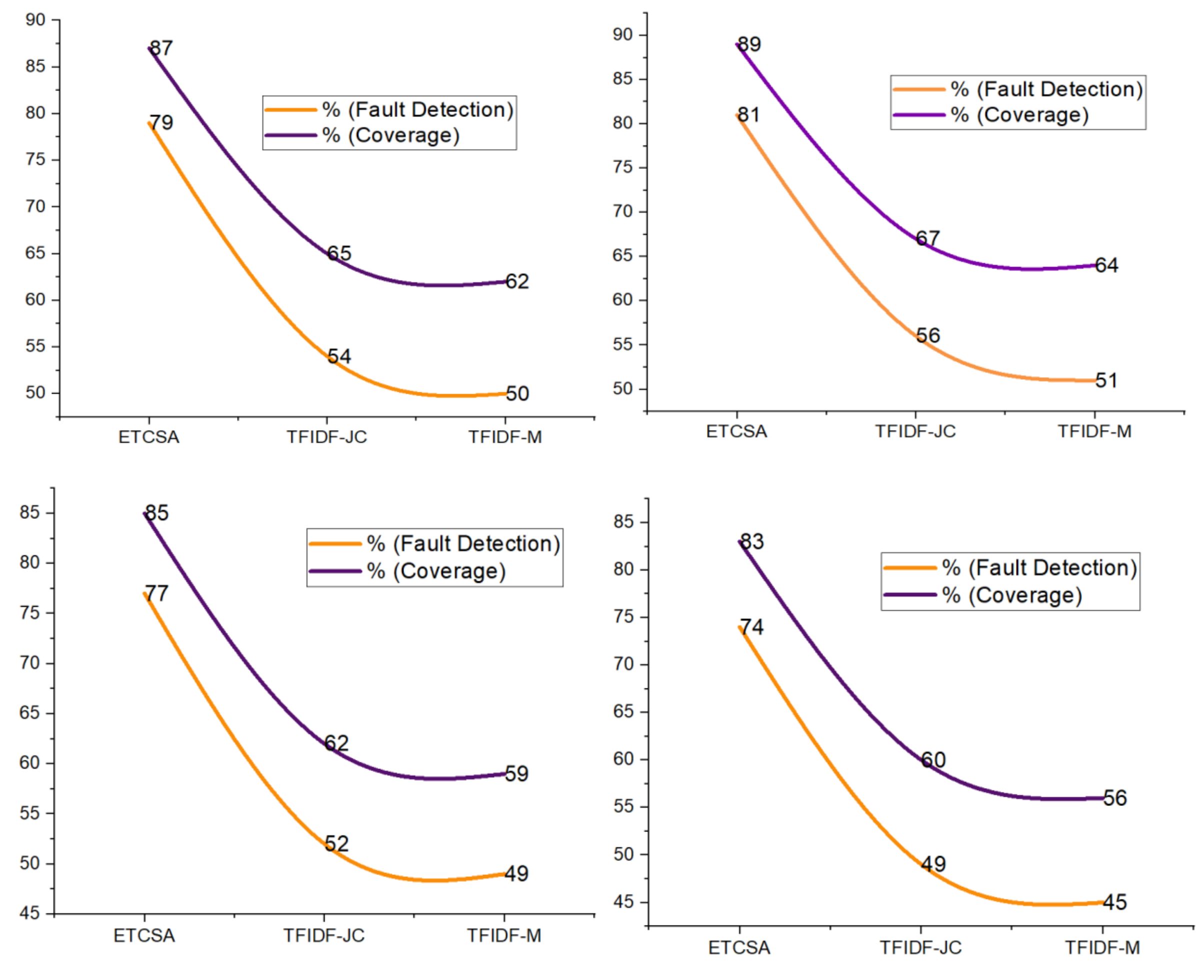

Figure 5. Fault & coverage ratio when test cases = (5. 10, 15, 20)

Conclusion

The primary objective of the Hybridized Framework ETCPM is to identify and select only those modified test cases that play a crucial role in determining quality parameters like time, cost, and efficiency. This approach aims to enhance regression testing techniques by excluding redundant and faulty test cases, thereby reducing the overall test size. The proposed hybrid method offers several advantages, including a decrease in execution time and an enhancement in fault detection capabilities. As for future research directions, there is a need for an improved regression testing approach that places more emphasis on performance, accuracy, security, and the reusability of regression testing processes.

Acknowledgment

This work was supported in part by the Research Management Center (RMC), Universiti Teknologi Malaysia (UTM), and the Ministry of Higher Education Malaysia (MOHE) through the UTM High Impact Research (UTMHR) grant scheme under (Vot number Q.J130000.2451.08G55).

Authors’ Declaration

- Conflicts of Interest: None.
- We hereby confirm that all the Figures and Tables in the manuscript are ours. Furthermore, any Figures and images, that are not ours, have been included with the necessary permission for re-publication, which is attached to the manuscript.
- Authors sign on ethical consideration’s approval.
- Ethical Clearance: The project was approved by the local ethical committee in University of Teknologi Malaysia Johor Bahru approved the project.

Authors’ Contribution Statement

M.A. S. Methodology, Code implementation, Writing.W. M.N. W.K: Supervision, Conceptualization, Reviewing. Writing an original draft, J. A. Investigation, Reviewing. Editing, Supervision,N. I. Investigation, Editing.

References

1. Muhammad H., Imran G., Muhammad F. P., and Seung R. J. A comprehensive review on regression test case prioritization techniques for web services. KSII Trans. Internet Inf. Syst. 2020; Vol. 14:No.5 . https://doi.org/10.3837/tiis.2020.05.001
2. Khatibsyarbini, M., et al., Test case prioritization approaches in regression testing: A systematic literature review. Inf Softw Technol. 2018; 93: p. 74-93. https://doi.org/10.1016/j.infsof.2017.08.014
3. Z. Yuan, Y. Lou, M. Liu, S. Ding, K. Wang, Y. Chen, and X. Peng.No more manual tests? evaluating and improving chatgpt for unit test generation. arXiv preprint arXiv:2305.04207. 2023. https://doi.org/10.1145/3395363.3397383
4. Mukherjee R. and Patnaik K. A survey on different approaches for software test case prioritization. J. King Saud Univ. Comput. Inf. Sci. Oct.2018; S1319157818303616. https://doi.org/10.1016/j.jksuci.2018.09.005
5. M. I. Younis, “DEO: a dynamic event order strategy for T-way sequence covering array test data generation,” Baghdad Sci. J. 2020;vol. 17, no. 2: p. 575, May. https://doi.org/10.21123/bsj.2020.17.2.0575
6. Bukhsh FA, Bukhsh ZA, Daneva M. A systematic literature review on requirement prioritization techniques and their empirical evaluation. Comput. Stand. Interfaces. 2020;69:103389. https://doi.org/10.1016/j.csi.2019.103389
7. Younis, M.I., Alsewari, A.R.A., Khang, N.Y., Zamli, K.Z., CTJ: Input-output based relation combinatorial testing strategy using jaya algorithm. Baghdad Sci. J. 2020;17 (3): pp. 1002-1009. https://doi.org/10.21123/BSJ.2020.17.3(SUPPL.).1002
8. Rongqi Pan, Mojtaba Bagherzadeh, Taher A. Ghaleb, and Lionel Briand. 2022. Test case selection and prioritization using machine learning: A systematic literature review. Empir. Softw. Eng. 2022;27, 2: 1–43. https://doi.org/10.1007/s10664-021-10066-6
9. Ali, S., et al., Enhanced regression testing technique for agile software development and continuous integration strategies. Softw. Qual. J. .2020; Vol. 28: p 397–423. https://doi.org/10.1007/s11219-019-09463-4
10. Pandey, A. and S. Banerjee.Test Suite Minimization in Regression Testing Using Hybrid Approach of ACO and GA. IJAMC. 2018; 9. https://doi.org/10.4018/IJAMC.2018070105
11. Agrawal, A.P. and A. Kaur, A comprehensive comparison of ant colony and hybrid particle swarm optimization algorithms through test case selection, in

Data engineering and intelligent computing. 2018, Springer. p. 397-405. https://doi.org/10.1007/978-981-10-3223-3_38
12. Chen et al., Chen J., Shang W., Shihab E., Perfjit: Test-level just-in-time prediction for performance regression introducing commits IEEE Trans. Softw. Eng. 2020; p. 1. https://doi.org/10.1109/TSE.2020.3023955
13. Singhal, S.; Jatana, N.; Subahi, A.F.; Gupta, C.; Khalaf, O.I.; Alotaibi, Y. Fault Coverage-Based Test Case Prioritization and Selection Using African Buffalo Optimization. Comput. Mater. Contin. 2023; 74: 6755–6774.https://doi.org/10.32604/cmc.2023.032308
14. Q. Zheng, Z. Ou, L. Liu, T. Liu, A novel method on software structure evaluation, in: Proceedings of the 2nd IEEE International Conference on Software Engineering and Service, ICSESS '11, IEEE.2011; pp. 251–254. https://doi.org/10.1016/j.jss.2020.110539
15. Magalhães, C., et al., HSP: A hybrid selection and prioritisation of regression test cases based on information retrieval and code coverage applied on an industrial case study. J. Syst. Softw. 2020; 159: p. 110430. https://doi.org/10.1016/j.jss.2019.110430

إطار عمل هجين لاستبعاد حالات الاختبار المشابهة والمعيبة في اختبار الانحدار

محمد عاصم صديق، وان م.ن. وان قادر، جوهانا أحمد، نوريني إبراهيم

كلية الحاسبات، الجامعة التكنولوجية، جوهور باهرو، ماليزيا.

الخلاصة

يعد اختبار الانحدار مرحلة حاسمة في تطوير البرامج التي تضمن أن التغييرات أو التحديثات الجديدة لنظام البرامج التي لا تؤدي إلى حدوث عيوب أو تؤثر سلبًا على الوظائف الحالية.
ومع ذلك، مع زيادة تعقيد أنظمة البرمجيات، يمكن أن تصبح كمية حالات الاختبار في مجموعة الانحدار كبيرة، مما يؤدي إلى زيادة وقت الاختبار واستهلاك الموارد. بالإضافة إلى ذلك، فإن وجود حالات اختبار زائدة عن الحاجة ومعيبة يمكن أن يزيد من إعاقة فعالية إجراء اختبار الانحدار.
ولمواجهة هذه التحديات، تقدم هذه الدراسة إطارًا هجينًا جديدًا لاستبعاد حالات الاختبار المشابهة والمعيبة في اختبار الانحدار ETCPM . يستفيد إطار العمل من تقنيات تحليل التعليمات البرمجية الآلية وبيانات تنفيذ الاختبار التاريخي لتحديد وإزالة حالات الاختبار المتكررة والمتشابهة والمعيبة من مجموعة الانحدار. توضح النتائج التجريبية أن إطار عمل ETCPM يقدم فوائد كبيرة في تقليل وقت الاختبار، وتحسين تخصيص الموارد، وتعزيز الجودة الشاملة لمجموعة اختبار الانحدار. من خلال الاستبعاد الذكي لحالات الاختبار المماثلة والمعيبة، يعمل ETCPM على تمكين فرق تطوير البرمجيات من تحقيق اختبار انحدار أسرع وأكثر موثوقية، مما يؤدي إلى تسريع دورات تسليم البرامج وتحسين رضا المستخدم النهائي..

الكلمات المفتاحية: اختبار الانحدار، تحديد أولويات حالة الاختبار، اختيار حالة الاختبار، جودة البرمجيات، صيانة البرمجيات.