International Journal of Engineering Trends and Technology
ISSN: 2231–5381 / https://doi.org/10.14445/22315381/IJETT-V73I9P118

Volume 73 Issue 9, 202-206, September 2025

Original Article

A Multi-Coverage Analysis Tool for Test Case Prioritization (TCP)

Mustafe Abdirashid Musse[1], Jamal Abdullahi Nuh[2], Salmi Baharom[1], Mohamed Abdullahi Ali[3,4], Abdihak Ahmed Abdullahi[3]

[1]*Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, Serdang, Selangor, Malaysia.*
[2]*Faculty of Information Sciences and Engineering, Management and Science University, Shah Alam, Malaysia.*
[3]*Faculty of Engineering and Technology, Salaam University, Mogadishu, Somalia.*
[4]*Department of Software Engineering, Faculty of Computer Science and Information Technology, Hormuud University, Mogadishu, Somalia.*

[2]*Corresponding Author : jamal_abdullahi@msu.edu.my*

Received: 15 April 2025 Revised: 28 August 2025 Accepted: 13 September 2025 Published: 30 September 2025

Abstract - *Regression testing is a crucial process for verifying that changes in the code do not introduce new faults. One widely used and effective technique to enhance regression testing is Test Case Prioritization (TCP), which focuses on determining the optimal order of test case execution to improve the rate of fault detection, especially in the early stages of testing. TCP tools are typically based on either single or multiple coverage criteria. However, tools based on single coverage criteria are often inefficient for regression testing, as they require repeated testing for each criterion, leading to increased time and cost. On the other hand, multiple coverage TCP tools offer a more comprehensive approach; they often lack empirical evidence regarding the most effective combinations of coverage criteria to use concurrently. To address these challenges, this study proposes a novel test environment for TCP and empirically compares various combinations of coverage criteria. A case study was conducted to evaluate the proposed environment. The results demonstrate its practical feasibility and improved effectiveness in enhancing the regression testing process.*

Keywords - *TCP, Multi-coverage, Software Testing.*

1. Introduction

Software testing is the action taken when we need to figure out the completeness, accuracy and productivity of a software by inspecting the behavior of the software to determine if what the user needed, and the product made are the same and their differences [2]. Regression testing is a crucial phase in the testing process during the SDLC because it certifies that the updates made to the software did not interfere with the other parts of the software to prevent unwanted behaviors [3]. The number of test suites used throughout the testing process increases as the development goes on, so it will cost too much to execute all the suites for every modification that is made. Moreover, regression testing checks that current features still work after changes have been made to the system to ensure that no new bugs have been added [11, 12]. Test case prioritization is a useful and productive approach that can be used to minimize the cost and time of the testing activities by organizing the test cases and detecting the faults earlier [6]. Many companies worldwide, such as Microsoft and Google, have recently used these techniques to make testing activities easier. Many test case prioritization techniques exist, such as coverage-based, requirement-based and change-based techniques [10]. There are several existing TCP tools that have been based on both single and multiple coverages. The single coverage TCP tools are not efficient for regression testing. Consequently, testing should be conducted multiple times for each coverage criterion, and this will result in time and cost [2]. Moreover, the multiple coverage TCP tools do not have empirical evidence on which of the coverage combinations are the most effective ones that can be used at the same time. Hence, in this study, a test environment for TCP has been proposed, and the combinations of coverage criteria have been empirically compared. Furthermore, a case study has been selected to evaluate the proposed TCP environment. The evaluation shows the practical feasibility and effectiveness of the TCP environment.

The remainder of the paper is organized as follows: Section 2 gives a literature review on the existing TCP tools. Section 3 presents the proposed TCP environment. Section 4 provides for the evaluation of the proposed TCP environment. Section 5 discusses this work. Section 6 gives the conclusion of the study.

2. Literature Review

Early research indicated that one of TCP's most important factors was coverage. According to the scope of code elements (such as statements or branches) that test cases cover, Rothermel et al. [13] provided a set of approaches that include total and additional coverage priority. Tests that cover the greatest ground overall are prioritized by total coverage, whereas tests that cover elements not covered in earlier runs are selected by additional coverage. These results were validated by real-world studies by Elbaum et al. [14], which demonstrated that using a single criterion, such as statement coverage, improved defect detection rates. Sinaga [15] developed a branch coverage-based TCP technique where test cases are arranged according to their ability to test various branches. The approach repeatedly selects tests covering the least unknown branches using a greedy algorithm. The Average Percentage of Faults Detected (APFD) on benchmark programs has been demonstrated to increase by 20–30%.

The novel test case prioritizing approach introduced by Alazzam and Nahar [7] incorporates both the line of code coverage metrics and the percentages of method coverage. They discovered that the more weighted test cases had a better chance of identifying flaws in the early stages of the testing procedure. The calculation was done by dividing the total number of methods and lines of code by the sum of the methods and lines of code covered combined. Srisura and Lawanna [8] proposed a method for choosing the generated false test cases during regression testing, and they did a systematic experiment to evaluate the quality of the method. A TCP technique called PORT was introduced by Prakash and Gomathi [2] and primarily concentrates on four factors: implementation complexity, fault proneness, customer priority, and requirements volatility of each demand. Additionally, they conducted multiple post hoc studies and an academic feasibility assessment.

To determine which code is prone to errors and modify the coverage-based TCP techniques to account for the code's defects, Wang and Tan [4] reviewed some of the current code testing methodologies to propose a new TCP methodology dubbed QTEP. Alemerien and Magel [15] conducted an experiment in another study to verify that the code coverage measures' values are consistent across branch, line, method, and statement.

Single-coverage analysis is straightforward and has been demonstrated to be effective in identifying faults in TCP; nevertheless, it has significant issues with scalability, the extent of fault identification, tool support, and metric consistency. More empirical testing, better hybrid measurements, algorithm optimization, and stronger tools are needed to ensure that single-coverage TCP remains effective in complex, current software development environments. Single-Coverage Analysis's Drawbacks and Difficulties for Test Case Prioritization:

- The limited scope of single-coverage measurements, like branch or statement coverage, sometimes leaves out intricate program relationships and interactions.
- Scalability and Computational Overhead: Approaches such as Sinaga's greedy approach for branch coverage and Alazzam and Nahar's method that integrates method and line coverage are computationally demanding for extensive test suites or intricate systems. The iterative process of prioritizing more coverage, as highlighted by Rothermel et al., intensifies this problem in large-scale applications.
- Validation and Generalizability: Empirical validations, including those conducted by Elbaum et al., Sinaga, and Prakash and Gomathi, are frequently confined to benchmarks or scholarly case studies. Real-world industrial applications, as observed by Srisura and Lawanna, are hardly examined, prompting inquiries over generalizability.
- Limited Fault Detection Scope: While single-coverage techniques, such as those by Elbaum et al. and Sinaga, improve the Average Percentage of Faults Detected (APFD), they may miss faults not directly tied to the chosen coverage metric. For example, Wang and Tan's QTEP technique adjusts for fault-proneness but still focuses on code-based coverage, potentially neglecting faults in non-code elements (e.g., configuration or integration issues).

This section discusses the literature on the existing TCP tools, shows that none of the reviewed TCP tools have addressed ranking test cases, and provides evidence on which coverage combinations are effective.

Therefore, this paper has been turned to propose a TCP environment that tackles the limitations of the previous TCP tools.

3. The Proposed TCP Environment

In this section, we have shown the proposed TCP environment and how the environment works. To use our TCP environment, the tester can follow these eight steps.

Step 1: Based on our tool, first, the user will visit a browser with the address below.

Fig. 1 Browser visit

Step 2: After that, the front end of the tool will appear as follows.

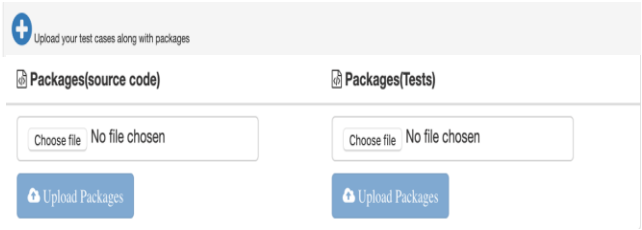


Fig. 2 Choose source code and test case

Step 3: The user will then upload both the source code and the test cases to be tested.

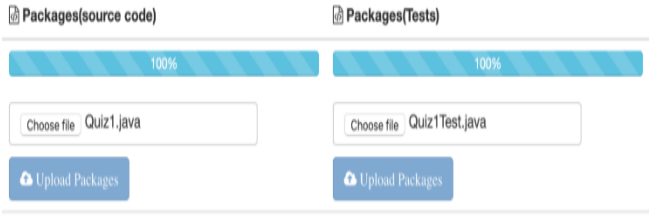


Fig. 3 Upload source code and test case

Step 4: After the upload process is complete, the user will then select which of the coverages they want to test.

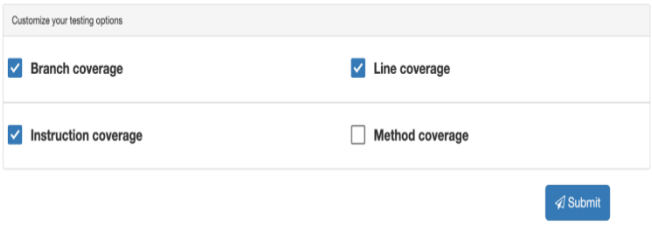
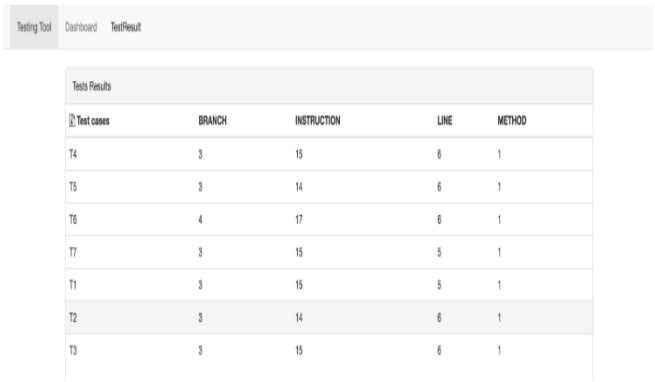


Fig. 4 Select coverages

Step 5: After the selection, they will then process all the requests from the user and generate the next table that shows the coverage values of the test cases.



Test cases	BRANCH	INSTRUCTION	LINE	METHOD
T4	3	15	6	1
T5	3	14	6	1
T6	4	17	6	1
T7	3	15	5	1
T1	3	15	5	1
T2	3	14	6	1
T3	3	15	6	1

Fig. 5 View coverage values

Step 6: The system generates the next figure that shows the criteria value, the priority value, and the weight and average percentages of the test cases.

Tests Criteria

Test cases	BRANCH	INSTRUCTION	LINE	METHOD	Priority value
T4	7.5	8.823529411764707	10	10	9.194165471066476
T5	7.5	8.235294117647058	10	10	9.055475848876135
T6	10	10	10	10	10
T7	7.5	8.823529411764707	8.333333333333334	10	8.747010999521759
T1	7.5	8.823529411764707	8.333333333333334	10	8.747010999521759
T2	7.5	8.235294117647058	10	10	9.055475848876135
T3	7.5	8.823529411764707	10	10	9.194165471066476
Average %	74.28571428571429	82.85714285714286	94.28571428571428	100	
Weight	0.21138211382113822	0.2357723577235772	0.2682926829268293	0.2845528455284553	

Fig. 6 View criteria value, priority value, average percentage and weight

Step 7: Then the system shows the user each test case and its priority value.

Tests Rank

test case	rank
T4	9.194165471066476
T5	9.055475848876135
T6	10
T7	8.747010999521759
T1	8.747010999521759
T2	9.055475848876135
T3	9.194165471066476

Fig. 7 View test case values

Step 8: Lastly, the rank of the test cases is shown to the user.

Test cases
T6
T4
T3
T5
T2
T7
T1

Fig. 8 View test case ranking

4. Evaluations of the Proposed TCP Environment

To evaluate the effectiveness of the proposed Test Case Prioritization (TCP) environment, a case study was conducted using a simple Java program designed to determine the largest of three input numbers. This program was chosen because it provides a straightforward yet non-trivial example, with multiple possible execution paths depending on the input values.

```
package user.uploaded.code;

public class Quiz1 {
    public static String QStatus(int a, int b, int c )
{       int status = -1;
        if(a > b && a > c ){
            status = 1;
        }else if (b > c){
            status = 2;
        }else{
            status =3;
        }
        switch (status){
            case 1 :
                return "a is the greatest";
            case 2 :
                return  "b is the greatest";
            case 3 :
                return "c is the greatest";
            default:
                return  "can not be determined";
        }
    }
}
```

Fig. 9 Case study

For this evaluation, seven carefully designed test cases were created to validate the correctness of the program under different input conditions, including:

1. All numbers being equal.
2. Two numbers are equal and greater than the third.
3. One number is clearly larger than the other two.
4. Boundary values such as negative inputs and zero.

The Java code and corresponding test cases were uploaded to the TCP environment. The system was then executed under multiple coverage criteria, including statement coverage, branch coverage, and path coverage.

```
package user.uploaded.code;


import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;
import static org.junit.jupiter.api.Assertions.assertEquals;

public class Quiz1Test {
    @Test
    public  void QStatutsTest(){
      assertEquals(Quiz1.QStatus(3,1,1),"a is the greatest");
    }
@Test
    public  void QStatutsTest(){
        assertEquals(Quiz1.QStatus(1,3,2),"b is the greatest");
    }
    @Test
    public  void QStatutsTest(){
      assertEquals(Quiz1.QStatus(1,2,3),"c is the greatest");
    }
    @Test
    public  void QStatutsTest(){
      assertEquals(Quiz1.QStatus(2,3,3),"c is the greatest");
    }
@Test
    public  void QStatutsTest(){
        assertEquals(Quiz1.QStatus(2,3,1),"b is the greatest");
    }
    @Test
    public  void QStatutsTest(){
        assertEquals(Quiz1.QStatus(3,1,2),"a is the
greatest");
    @Test
    public  void QStatutsTest(){
        assertEquals(Quiz1.QStatus(2,1,2),"c is the
greatest");
}
```

Fig. 10 Test cases

The TCP environment automatically processed the test suite and generated the following outputs:

1. A coverage matrix mapping each test case against the program's statements and branches.
2. A table of computed criteria values, weights, and the corresponding average percentages.
3. A final prioritized ranking table showing the execution order of the test cases based on their effectiveness.

This evaluation highlighted the ability of the TCP environment to:

a) Combine multiple coverage criteria into a weighted prioritization strategy.
b) Demonstrate how certain test cases achieve higher coverage efficiency.
c) Provide actionable insights into optimizing the execution order of tests for faster fault detection.

Overall, the case study confirmed that the proposed TCP environment can produce meaningful prioritization results while offering flexibility in selecting and combining coverage criteria. This ensures more efficient software testing, especially in larger systems where test execution costs are significant.

5. Discussions

Several TCP tools have been proposed in the past literature. These tools have been ignored to show evidence to the tester which of the coverage combinations are effective to be used at the same time, and they did not show the ranking of test cases based on coverage value. So, the tester decides on which test case to run first. Moreover, most of the previous studies were based on single coverage, which was time-consuming. Furthermore, multiple coverage TCP tools do not have empirical evidence on which of the coverage combinations are the most effective ones that can be used at the same time. Unlike the previous studies, we have addressed the issues of the previous studies by proposing a TCP environment. We demonstrated how to rank the test cases and evidence on effective coverage combinations by using the proposed TCP environment. We have conducted a case study to evaluate practically how the proposed TCP environment is effective. We conclude that the proposed environment will significantly reduce TCP time and cost and increase productivity.

6. Conclusion

Prioritizing the test cases is an effective and broadly used technique to carry out regression testing or to identify errors at the early stages of the testing process. TCP concerns basic planning of the test case execution, aiming to enhance the success of the software testing process by raising the rate of fault detection. Therefore, several TCP tools have been

proposed based on both single and multiple coverages. The single coverage TCP tools are not efficient for regression testing. Consequently, testing should be conducted multiple times for each coverage criterion, and this will result in time and cost consumption. In addition, the multiple coverage TCP tools do not have empirical evidence on which of the coverage combinations are the most effective ones that can be used at the same time. Nevertheless, a TCP environment has been proposed for prioritizing test cases based on multiple coverage criteria. Additionally, a case study has been conducted to evaluate the practical feasibility and effectiveness of the proposed TCP environment. In short, the proposed environment will significantly reduce TCP time and cost and increase productivity. The limitations of this study include the fact that test case results cannot be exported to different file formats, such as PDF, and the proposed TCP environment is limited to testing a few lines of code. Future work: The environment will be improved for exporting different file formats. Moreover, it can expand the proposed environment to accept large amounts of code.

References

[1] Isha Sharma, Jasleen Kaur, and Manisha Sahni, "A Test Case Prioritization Approach in Regression Testing," *International Journal of Computer Science and Mobile Computing*, vol. 3, no. 7, pp. 607-614, 2014. [Google Scholar] [Publisher Link]

[2] Iyad Alazzam, and Khalid M. O Nahar, "Combined Source Code Approach for Test Case Prioritization," *Proceedings of the 2018 International Conference on Information Science and System*, Jeju Republic of Korea, pp. 12-15, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[3] Zan Wang et al., "Improved Annealing-Genetic Algorithm for Test Case Prioritization," *Computing and Informatics*, vol. 36, no. 3, pp. 705-732, 2017. [Google Scholar] [Publisher Link]

[4] Song Wang, Jaechang Nam, and Lin Tan, "QTEP: Quality-Aware Test Case Prioritization," *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, Paderborn Germany, pp. 523-534, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[5] N. Prakash, and K. Gomathi, "Improving Test Efficiency through Multiple Criteria Coverage based Test Case Prioritization," *International Journal Science Engineering Research*, vol. 5, no. 4, pp. 430-435, 2014. [Google Scholar]

[6] Khalid Alemerien, and Kenneth Magel, "Examining the Effectiveness of Testing Coverage Tools: An Empirical Study," *International Journal of Software Engineering and its Applications*, vol. 8, no. 5, pp. 139-162, 2014. [Google Scholar] [Publisher Link]

[7] Benjawan Srisura, and Adtha Lawanna, "False Test Case Selection: Improvement of Regression Testing Approach," *2016 13th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, Chiang Mai, Thailand, pp. 1-6, 2016. [CrossRef] [Google Scholar] [Publisher Link]

[8] D.R. Medhun Hashini, and B. Varun, "Clustering Approach to Test Case Prioritization using Code Coverage Metric," *International Journal of Engineering and Computer Science*, vol. 3, no. 4, pp. 5304- 5306, 2014. [Google Scholar] [Publisher Link]

[9] Alessandro Marchetto et al., "A Multi-Objective Technique to Prioritize Test Cases," *IEEE Transactions on Software Engineering*, vol. 42, no. 10, pp. 918-940, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[10] Dharmveer Kumar Yadav, and Sandip Dutta, "Regression Test Case Prioritization Technique using Genetic Algorithm," *Advances in Computational Intelligence*, Ranchi, India, pp. 133-140, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[11] Paul Ammann, and Jeff Offutt, *Introduction to Software Testing*, Cambridge University Press, 2016. [CrossRef] [Google Scholar] [Publisher Link]

[12] G. Rothermel, and M.J. Harrold, "Analyzing Regression Test Selection Techniques," *IEEE Transactions on Software Engineering*, vol. 22, no. 8, pp. 529-551, 1996. [CrossRef] [Google Scholar] [Publisher Link]

[13] Gregg Rothermel et al., "Prioritizing Test Cases for Regression Testing," *IEEE Transactions on Software Engineering*, vol. 27, no. 10, pp. 929-948, 2001. [CrossRef] [Google Scholar] [Publisher Link]

[14] S. Elbaum et al., "Test Case Prioritization: A Family of Empirical Studies," *IEEE Transactions on Software Engineering*, vol. 28, no. 2, pp. 159-182, 2002. [CrossRef] [Google Scholar] [Publisher Link]

[15] Arnaldo Marulitua Sinaga, "Branch Coverage Based Test Case Prioritization," *ARPN Journal of Engineering and Applied Sciences*, vol. 10, no. 3, pp. 1131-1137, 2015. [Google Scholar] [Publisher Link]