ISSN 2250-3153

"Reduction in Test Cases using Regression testing approach and cost effective test prioritization testing techniques" - APFD measure

Rimmi Saini[1], Satinder Saini[2], Deepa Gupta[3], Ajay Rana[4]

[1]Department of Computer Science and Engineering
Greater Noida Group of Institutions
Knowledge Park II,
Greater Noida 201306

[2]Department of Computer Science and Engineering
Lovely Campus, Lovely Professional University
Phagwara, Jalandhar, Punjab (India)-144402

[3]Department of Computer Science and Engineering
Amity Institute of Information Technology
Amity University Campus,
I 1 Block, 3rd Floor, Sector -125,
Noida 201303

[4]Department of Computer Science and Engineering
Amity School of Engineering and Technology
Amity University, Sec-125, Noida 201303

Abstract- Testing of software is an important means of assessing the software to determine its quality. Software testing has proven that testing, analysis, and debugging cost usually consume over 50% of the costs associated with the development of large software systems. Software testing and retesting occurs continuously during the software development lifecycle to detect errors as early as possible. The sizes of test suites grow as software evolves. Regression testing is an expensive testing process used to validate modified software and detect whether new faults have been introduced into previously tested code. To reduce the cost of regression testing, software testers may prioritize their test cases such that those which are more important, by some measure, are run earlier in the regression testing process. Test case prioritization techniques schedule test cases for regression testing in an order that increases their ability to meet some performance goal. One potential goal of such prioritization is to increase a test suite's rate of fault detection. An improved rate of fault detection during testing can provide faster feedback on the system under test and let software engineers begin correcting faults earlier than might otherwise be possible. Test case prioritization techniques schedule test cases for execution in an order that attempts to increase their effectiveness at meeting some performance goal. Sometimes it is difficult to choose appropriate prioritization technique for a given testing scenario. Thus we analyse the fault detection rates. This paper presents several strategies for selecting a smaller number of test suites by reordering the test cases. We have used a metric --- APFD for measuring rate of fault detection. Here we have reordered the test cases, applied APFD metric to various test suites and reported the results of experiments using those techniques. The rate of fault detection is measured for different reordered test cases, comparing the rates achieved by randomly ordered and optimally ordered test cases.

Index Terms- software testing, test case, test prioritization, tests case reordering, test suits, regression testing

I. INTRODUCTION

Testing of software is an important means of assessing the software to determine its quality. Software Testing is an important process that is performed to support quality assurance or we can say testing of software is an important means of assessing the software to determine its quality. Testing activities support quality assurance by gathering information about the nature of the software being studied. Software testing has been proven that testing, analysis, and debugging costs usually consume over 50% of the costs associated with the development of large software systems. These activities consist of designing test cases, executing the software with those test cases, and examining the results produced by those executions. Studies shows that testing consumes 40~50% of development efforts, and consumes more effort for systems that require higher levels of reliability. So it is a significant part of the software engineering.

Software testing is a very broad area, which involves many other technical and non-technical areas, such as specification, design and implementation, maintenance, process and management issues in software engineering.
The definition of testing varies according to the purpose, process, and level of testing described. While discussing about software testing, let give an overview of software development life cycle (or SDLC) and a general software testing process. In general, the SDLC process contains the following phases: requirement gathering, design & analysis, development, testing, and Maintenance [1]. These phases can be represented as follows:

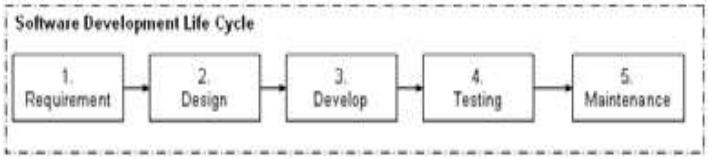


Figure-1: A General SDLC Process

One of the phases of SDLC is the testing phase [5] which contains the following processes:

- Test planning,
- Test development,
- Test execution
- Evaluation of result

These processes can be represented as follows:

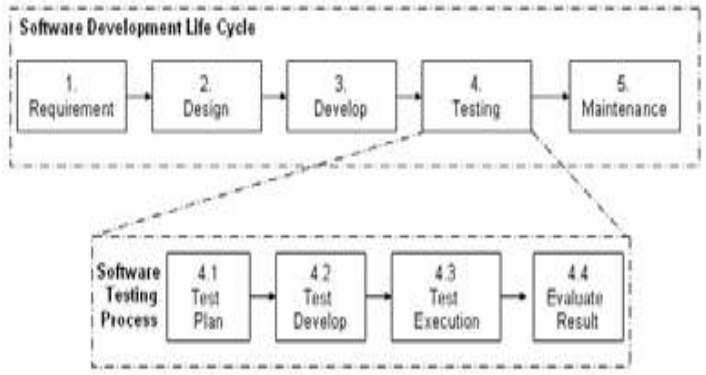


Figure-2: A General Software Testing Process

Miller gives a good description of testing in [2]:

"The general aim of testing is to affirm the quality of software systems by systematically exercising the software in carefully controlled circumstances".

Software Testing is the process of executing a program or system with the intent of finding errors [3]. Software testing is a comprehensive set of activities which is conducted with the intent of finding the errors in software. In the software development process, software testing is the one activity which aimed at evaluating a software item such as system, subsystem and features (e.g. functionality, performance and security) against a given set of system requirements. Also we can say that a software testing is the process of validating and verifying that a program functions properly.
The researchers in the field of software testing have proven that software testing is one of the most critically important phases of the software development life cycle, and it consumes significant resources in terms of effort, time and cost.

Arden [35] said that "The impact of software errors is enormous because virtually every business in the United States now depends on software for the development, production, distribution, and after sales support of products and services. Innovations in fields ranging from robotic manufacturing to nanotechnology and human genetics research have been enabled by low-cost computational and control capabilities supplied by computers and software."

In a study conducted by NIST in 2002, reports that software bugs costs the economy the most. More than one-third of this cost could be avoided if better software testing was performed [35]. According to Boris [36], software testing takes around 40-70% of the time and cost of the software development process. Numerous approaches have been proposed to reduce time and cost during software testing process. These approaches include test case prioritization techniques and test case reduction techniques. Various Test case prioritization techniques and Test case reduction techniques are explained in [37], [38], [39], [40], [41], [16], [17] and [19]. Many empirical studies for prioritizing test cases have also been conducted and explained in [10], [11], [12], [13], [14] and [15].

Gregg Rothermel [39] has proven that prioritizing and scheduling test cases are one of the most critical tasks during the software testing process. In an industrial collaborators report, Gregg Rothermel referred that if there are approximately 20,000 lines of code then running the entire test cases requires seven weeks. In such a situation, test engineers may want to prioritize and schedule test cases in order such that test cases with higher priority are executed first. In the research of Gregg Rothermel [37] [9], he stated that test case prioritization methods and process are very much required because of the following:
(a) The regression testing phase consumes a lot of time and cost to run
(b) There is not enough time or resources to run the entire test suite
(c) There is a need to decide which test cases to run first.

There are various goals to increase the effectiveness: one such goal involves rate of fault detection. Fault detection is a measure of how quickly faults are detected within the testing process. During testing an improved rate of fault detection can provide faster feedback on the system under test and help the software engineers to begin correcting faults as earlier as possible. One of the application of prioritization techniques is regression testing. It is defined as the retesting of software following some modifications. So here prioritization techniques can take advantage of information gathered about the previous execution of test cases to obtain test case orderings. In order to reduce the cost of regression testing, software testers may prioritize their test cases so that those test cases which are more important are run earlier in the regression testing process. One of the potential goals of such type prioritization is to increase a test suite's rate of fault detection.

Test case prioritization techniques prioritize and schedule test cases in an order that attempts to maximize some objective function. For example, software test engineers might wish to schedule test cases in an order that achieves code coverage at the fastest rate possible, exercises features in order of expected frequency of use, or exercises subsystems in an order that reflects their historical propensity to fail [4]. When the time required to execute all test cases in a test suite is short, test case prioritization may not be cost effective rather it will be most expedient simply to schedule test cases in any order [37], [9]. When the time required to run all test cases in the test suite is sufficiently long, the benefits offered by test case prioritization methods become more significant. As we know test case prioritization methods have great benefits for software test engineers but still outstanding major research issues that should be addressed [4]. The major areas of research issues are related to (a) existing test case prioritization methods ignore the practical weight factors in their ranking algorithm (b) existing techniques have an inefficient weight algorithm and (c) those techniques are lack of the automation during the prioritization process [30].

Section 2 discusses a literature review of existing test case prioritization techniques and prioritization processes. Section 3 proposes research challenges measuring the effectiveness. Section 3 also proposes test case prioritization techniques schedule test cases for execution in an order that attempts to increase their effectiveness to meet some performance goal. Section 4 provides the comparison related to APFD metric and calculating the rate of fault detection. Section 5 discusses conclusion and future work for test case prioritization area. The last section represents all source references used in this paper.

II. LITERATURE SURVEY

Software testing has been widely used as a way to help software engineers to develop high-quality systems. It is one of the important process that is performed to support quality assurance by gathering information about the nature of the software being studied M. J. Harrold [34]. These activities consist of designing test cases, executing the software with those test cases, and then examining the results produced by those executions. Boris [36] indicates that more than fifty percent of the cost of software development is devoted to testing with the percentage for testing critical software being even higher. As more as the software becomes pervasive, is used more often to perform critical tasks and the importance of its quality will remain high. When the engineers are unable to find efficient ways to perform effective testing, the percentage of development costs devoted to testing may increase significantly.

Test techniques include the process of executing a program or application with the intent of finding software bugs. It can also be stated as the process of validating and verifying that software meets the business and technical requirements that guided its design and development, so that it works as expected [30]. Software testing can be implemented at any time in the development process. Software engineers generally save test suites that they develop so that they can easily reuse those suites later as the software evolves. Reusing test cases in regression testing process is pervasive in the software industry [34] and can save as much as one-half of the cost of software maintenance [36].

Arup Abhinna Acharya et. al., in their paper proposes a method to prioritize the test cases for testing component dependency in a Component Based Software Development (CBSD) environment using Greedy Approach [6]. According to Arup Abhinna Acharya et. al., the cost and time required for regression testing can be minimized by using the prioritization technique discussed in their paper. Here they have proposed a model based prioritization method by considering the number of Object Interactions per unit time as the objective function. Here more importance is given to number of inter component object interactions present because maximum faults are expected to be present when components interact with each other [6]. James A Jones et al. [7] have presented an algorithm for test suite prioritization that incorporates the complexities of modified condition or decision coverage.

Boris [8], claimed software testing should take around 40-70% of the time and cost of the software development process. Many approaches have been proposed to reduce time and cost during software testing process, including test case Prioritization techniques and test case reduction techniques. According to Elbaum et.al. , Test case prioritization techniques help engineers execute regression tests in an order that achieves testing objectives earlier in the testing process. In their paper they discussed that one testing objective involves *rate of fault detection* {a measure of how quickly a test order detects faults}. An improved rate of fault detection can provide earlier feedback on the system under test, enable earlier debugging, and increase the likelihood that, if testing is prematurely halted, those test cases that offer the greatest fault detection ability in the available testing time will have been executed [18].

Elbaum et. al. [22][23], Rothermel et. al. [20][21] chosen four heuristics for the target prioritization technique in their literature and investigation. Their literature and investigations in empirical studies that could or have been easily implemented by practitioners, allow them to examine two of the key dimensions of differences among techniques. These dimensions are the uses of feedback and information on modifications. For simplicity and to facilitate comparison, Elbaum et.al., Rothermel et al., restricted their attention to function-coverage-based techniques. Zheng Li et al. have tested experimentally that genetic algorithms perform well for test case prioritization [24].

In [32], Rothermel et al. pointed that the potential goal of prioritization is to increase a test suite's rate of fault detection earlier in the software process. It has been tested experimentally in [33] that some of the biggest causes for project failures are lack of user input and changing or incomplete requirements. Software engineers save the test cases and re-run the test cases as regression test in later versions.

Kim and Porter [25] also presented a technique, which they refer to as a "history-based prioritization" that utilizes information from previous testing cycles to select the test cases that must be

executed for a new version of the program. But this technique is not sometimes considered as a "prioritization technique" in the sense done in the literature because it imposes no ordering on test cases. Ordering of tests is the essential characteristic for the definition of prioritization. This approach selects a subset of a test suite, using history information to determine which test cases should be selected, and is more accurately described as a "regression test selection technique" by Rothermel and Harrold in 1996 [26].

Gregg Rothermel [27] has proven that prioritizing and scheduling test cases are one of the most critical tasks during the software testing process. They referred the industrial collaborators reports, which show approx 20,000 lines of code, running the entire test cases and require seven weeks. In this situation, test engineers may want to prioritize and schedule those test cases in order that those test cases with higher priority are executed first. Additionally, he [28], [29] stated that test case prioritization methods and process are required, because: (a) the regression testing phase consumes a lot of time and cost to run,
And (b) there is not enough time or resources to run the entire test suite (c) there is a need to decide which test cases to run first [30].
According to A. G. Malishevsky et al., Test case prioritization has been primarily applied to improve regression testing effects as mentioned in their paper [31].

As we have seen above, Rothermel [38], [39] gave an interesting example of industrial collaborators. In such cases, testers may want to order their test cases so that those test cases with the highest priority, according to some criterion, are run first. Test case prioritization techniques prioritize and schedule test cases in an order that attempts to maximize some objective function. For example, software test engineers wish to schedule test cases in an order that achieves the code coverage at the fastest possible rate, exercises features in order of expected frequency of use, or exercises subsystems in an order that reflects their historical propensity to fail [4]. One of the interesting feature in Test case prioritization is: When the time required to execute all test cases in a test suite is short, test case prioritization may not be cost effective - it may be most expedient simply to schedule test cases in any order [38], [9]. When the time required to run all test cases in the test suite is sufficiently long, the benefits offered by test case prioritization methods become more significant. Test case prioritization techniques provide a way to schedule and run test cases, which have the highest priority in order to provide earlier detected faults. This study presents numerous techniques developed, between 2002 and 2008, that can improve a test suite's rate of fault detection. With existing test case prioritization techniques researched, there is "4C" classification of those existing techniques, based on their prioritization algorithm's characteristics. This classification is given as follows:

1. Customer Requirement-based techniques.
2. Coverage-based techniques.
3. Cost Effective-based techniques.
4. Chronographic history-based techniques [30].

III. TEST CASE PRIORITIZATION PROBLEM

Rothermel et al. (2001) define the test case prioritization problem and describe several issues relevant to its solution. Now to understand testing techniques properly, we will study the test case prioritization techniques and its problem in detail. The test case prioritization problem is defined as follows:

In Test Case Prioritization Problem, we have been given:
$T \rightarrow$ a test suite;
$PT \rightarrow$ the set of permutations of T;
$f \rightarrow$ a function from PT to the real numbers.
The Problem of Test Case Prioritization is:
Find $T0\ 2\ PT$ such that $(8T00)$ $(T00\ 2\ PT)$ $(T00\ 6 = T0)$ $[f(T0), f(T00)]$.
Here, PT represents the set of all possible prioritizations (orderings) of T, and f is a function that is applied to any such ordering and yields an *award value* for that ordering [18].

There are many of the possible goals for prioritization. Our main focus is on increasing the likelihood of revealing faults as earlier as possible in the testing process. By this goal one can informally improve the test suite's *rate of fault detection*. And to quantify this goal, we will use a metric APFD (Average of the Percentage of Faults Detected), introduced by Rothermel et.al.

APFD helps to measures the weighted average of the percentage of faults detected over the life of the suite. The APFD values range from 0 to 100; the higher the number, the faster (better) is the fault detection rates.

Let T be a test suite containing n test cases, and let F be a set of m faults revealed by T.
Let TFi be the first test case in ordering $T0$ of T which reveals fault i.
The APFD for test suite $T0$ is given by the equation:

$$APFD = 1 + \frac{TF_1 + TF_2 + ... + TF_m}{nm} + \frac{1}{2n} \quad (1)$$

We will consider a program with a test suite of five test cases. These test cases are from V to Z. The program contains ten faults detected by those test cases, as shown by the table in Table-1.

Table-1: Test suite and faults exposed

Tests	Faults									
	1	2	3	4	5	6	7	8	9	10
V	X				X					
W						X	X			
X	X	X	X	X	X	X	X			
Y					X					
Z								X	X	X

IV. MEASURING EFFECTIVENESS

Test case prioritization techniques schedule test cases for execution in an order that attempts to increase their effectiveness to meet some performance goal. In measuring the effectiveness,

we measure how rapidly a prioritized test suite detects faults. In other words, for finding the rate of fault detection of the test suite, we require an appropriate objective function. Here we are using APFD (Average of the Percentage of Faults Detected) metric. APFD helps to measures the weighted average of the percentage of faults detected over the life of the suite. The APFD values range from 0 to 100; the higher the number, the faster (better) is the fault detection rates. As an example we have taken a program with test suite of 5 test cases. These test cases are named V to Z. The program contains 10 faults detected by those test cases. Table-1 shows test suite and faults exposed. The APFD values represent the area under the curve by plotting percentage of faults detected on the y-axis of a graph, and Test Suite Fraction on the x-axis of a graph. The area inside the inscribed rectangles represents the weighted percentage of faults detected over the corresponding fraction of the test suite. In other words, the area under the curve thus represents the weighted average of the percentage of faults detected over the life of the test suite, since our aim is to measure how quickly the percentage of faults detected over the life of the suite.

Now we will illustrate the APFD measure for different test case orders using graphs. Consider the following orders of test cases:
Order *T*1: V-W-X-Y-Z; Order *T2*: W-V-X-Y-Z; Order *T3*: X-Z-W-V-Y; Order *T*4: Z-Y-X-W-V; Order *T5*: Z-X-W-V-Y.

Suppose we place the test cases in order V-W-X-Y-Z to form prioritized test suite. The graph for order T1: V-W-X-Y-Z is shown as:

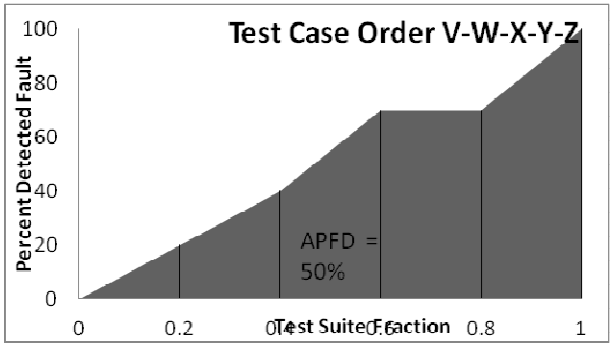


Figure 3A: APFD for prioritized test suite T1

Figure 3A shows the percentage of detected faults versus the fraction of test suites according to order T1. After running test case V, 2 of the 10 faults are detected; thus 20% of the faults have been detected. After running test case W, 2 more faults are detected and thus 40% of the faults have been detected. The area under the curve represents the weighted average of the percentage of faults detected over the life of the test suite. This area is the prioritized test suite's average percentage faults detected metric (APFD); the APFD is 50% in this example.

Suppose now we swap V with W and then place the test cases in order W-V-X-Y-Z to form prioritized test suite. The order T2: W-V-X-Y-Z received after swapping V and W alters the rate at which *particular* faults are detected, but not the overall rates of fault detection. Thus the orders V-W-X-Y-Z and W-V-X-Y-Z are equivalent in terms of rate of fault detection. This equivalence is reflected in equivalent APFDs (50%).

When the test case order is Order *T3*: X-Z-W-V-Y to form prioritized test suite. The graph for Order *T3*: X-Z-W-V-Y is shown as:

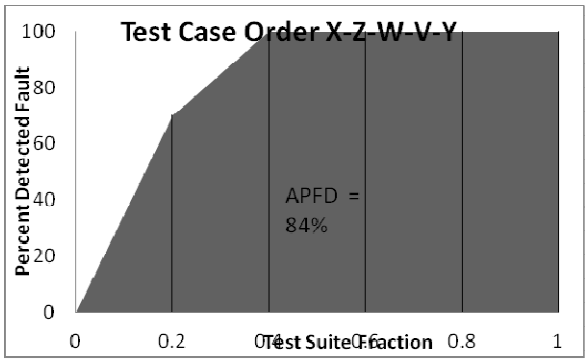


Figure 3B: APFD for prioritized test suite T3

Figure 3B also shows the percentage of detected faults versus the fraction of test suites according to order T3: X-Z-W-V-Y. Under the APFD metric this order is assigned an APFD value of 84%. This test order *T*3 (Figure 3B) is much faster detecting test order than *T*1 and order T2, (and in fact, an optimal order) with an APFD of 84.0%.

Now we consider the test cases in order Z-Y-X-W-V to form prioritized test suite. The graph for order *T*4: Z-Y-X-W-V (Figure 3C) is shown as:

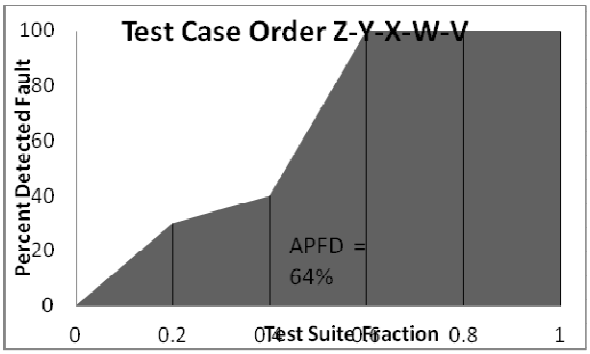


Figure 3C: APFD for prioritized test suite T4

Figure 3C also shows the percentage of detected faults versus the fraction of test suites according to order *T*4: Z-Y-X-W-V. Under the APFD metric this order is assigned an APFD value of 64%. This test order *T*4 (Figure 3C) is slower in detecting test order than *T3* and much faster detecting test order than *T*1 and order T2 with an APFD of 64.0%.

Consider the test cases in order Z-X-W-V-Y to form prioritized test suite. The graph for order *T5*: Z-X-W-V-Y (Figure 3D) is shown as:

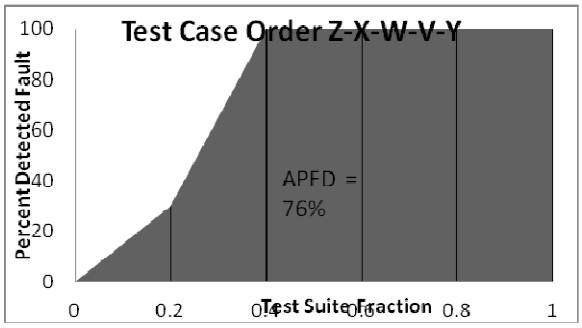


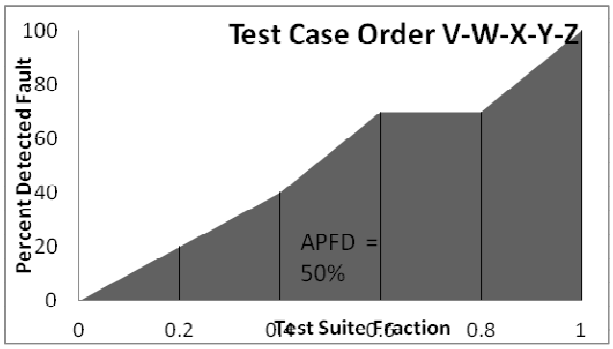
Figure 3D: APFD for prioritized test suite T5

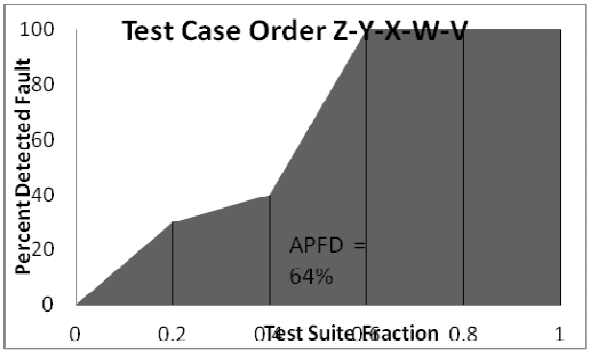
Figure 3D also shows the percentage of detected faults versus the fraction of test suites according to order *T5*: Z-X-W-V-Y. Under the APFD metric this order is assigned an APFD value of 76%.

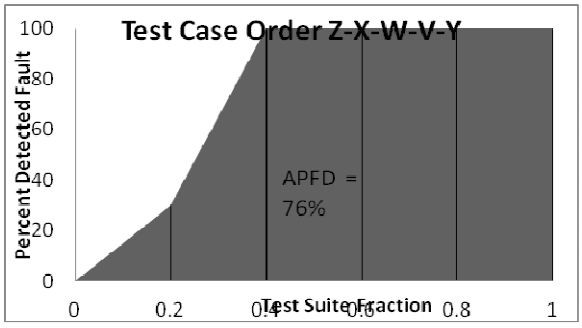
V. COMPARISON

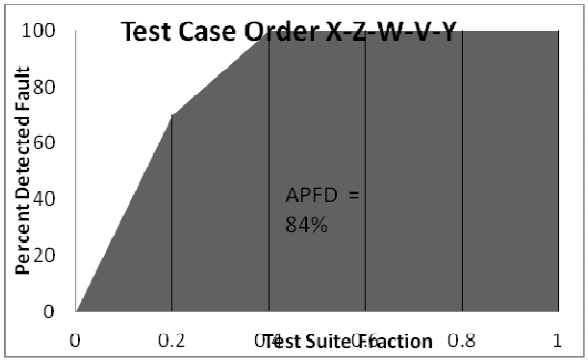
Software maintenance is an important and costly activity of the software development lifecycle. To ensure proper maintenance the software undergoes regression testing. As we know, it is very inefficient to re execute every test case in regression testing for small changes. Hence test case prioritization is a technique to schedule the test case in an order that maximizes some objective function. One such objective function involves rate of fault detection. The Rate of fault detection is defined as a measure of how quickly faults are detected within the testing process. Early fault detection can provide a faster feedback generating a scope for debuggers to carry out their task at an early stage. Test case prioritization techniques schedule test cases for execution in an order that attempts to increase their effectiveness to meet some performance goal. In our experiments while measuring the effectiveness, we measure how rapidly a prioritized test suite detects faults. Thus, for finding the rate of fault detection of the test suite, we require APFD (Average of the Percentage of Faults Detected) metric. APFD helps to measures the weighted average of the percentage of faults detected over the life of the suite. The APFD values range from 0 to 100; the higher the number, the faster (better) is the fault detection rates. As explained above we have taken a program with test suite of 5 test cases named V to Z and contain 10 faults detected by those test cases. Table-1 shows test suite and faults exposed. The percentage of fault detected is plotted on y-axis and test suite fraction on x-axis of graph. The APFD values represent the area under the curve. The area under the curve thus represents the weighted average of the percentage of faults detected over the life of the test suite, since our aim is to measure how quickly the percentage of faults detected over the life of the suite.

As a summery, the following graphs illustrate the APFD measure for different test case orders we considered above.









From the above graph we can make a table of comparison showing the test case orders and Average of the Percentage of Faults Detected. The comparison table is shown as:

S.No.	Test Case Order	APFD Measure (%)
1	T1: V-W-X-Y-Z	50%
2	T2: W-V-X-Y-Z	50%

3	T3: X-Z-W-V-Y	84%
4	T4: Z-Y-X-W-V	64%
5	T5: Z-X-W-V-Y	76%

Table-2: Calculation of % fault detection

From table-2 we can conclude that when we consider the test case order T3: X-Z-W-V-Y, the APFD is 84%. This means that area is the prioritized test suite's average percentage faults detected metric (APFD) is 84%. Thus Test Prioritization Testing Techniques are found to be very effective in early fault detection as compared to non-prioritize approach.

The APFD metric presented above relies on two assumptions: (1) all faults have equal severity, and (2) all test cases have equal costs. These assumptions simply plots the percentage of faults detected against the fraction of the test suite run. In general when these assumptions hold, the metric operates well but in practice there are cases in which this APFD metric do not hold and can give the unsatisfactory results.

VI. CONCLUSION

We have presented techniques for prioritizing test cases that attempt to account for the effects of varying test case fault severity. Our focus in this work has been the APFD metric. If we select the similar types of test cases which give the same expected results then we can find the percentage of fault detection very easily. The higher APFD value the more number of faults are detected at early stages of software development. So now the main emphasis should be given on the techniques which can reduce the cost of testing. Since cost is the factor which affect the software development the most. If we can predict the set of test suites or test cases at the early stages of the software development, then cost of the software testing can be reduced. Time is also another factor. When the time required to execute all test cases in a test suite is short, test case prioritization may not be cost effective - it may be most expedient simply to schedule test cases in any order. When the time required to run all test cases in the test suite is sufficiently long, the benefits offered by test case prioritization methods become more significant. Suppose all ten faults are equally severe. Test cases V, W, Y, and Z each require one hour executing, but test case X requires ten hours. Consider test case order X-Z-W-V-Y. Under the APFD metric this order is assigned an APFD value of 84% (Figure 3B). The alternative to this test case is order Z–X–W–V–Y. This order is assigned an APFD value of 76% (Figure 3D).This metric does not differentiate test cases in terms of relative costs. Here all vertical bars in the graph have the same width. The APFD for this order is 76% – lower than the score for test case order X-Z-W-V-Y. In terms of faults-detected-per-hour, the second order (Z–X–W–V–Y) is preferable since it detects 3 faults in the first hour, and remains better in terms of faults-detected-per-hour than the first order up through the end of execution of the second test case. We have kept the cost uniform while varying fault severities. So a new model should be generated in which cost must also be considered along with the rate of fault detection. The cost and time required for regression testing can be minimized by using the prioritization technique.

VII. CONTINUING WORK

We have presented APFD, for assessing the rate of fault detection of prioritized test cases that incorporates varying test case and fault costs. The proposed method can further be extended to prioritize test cases to perform regression testing for real time systems and distributed systems. We have presented a few prioritization techniques intended to improve prioritization under this metric, but there may be other techniques better suited to maximizing the objective function represented by APFD. Further we will try to construct an infrastructure to support experiments in which factors such as test cost and fault severity can be controlled and varied, and test cost and fault severity data drawn directly.

REFERENCES

[1] H. Agrawal, J. R. Horgan, E. W. Krauser, and S. London, "Incremental regression testing", IEEE International Conference on Software Maintenance, pages 348–357, 1993.

[2] E. F. Miller, "Introduction to Software Testing Technology," *Tutorial: Software Testing & Validation Techniques*, Second Edition, IEEE Catalog No. EHO 180-0, pp. 4-16 [Miller81]

[3] [Myers79] Myers, Glenford J., *The art of software testing,* Publication info: New York : Wiley, c1979. ISBN: 0471043281 Physical description: xi, 177 p. : ill. ; 24 cm.

[4] Journal of Theoretical and Applied Information Technology, © 2005 - 2010 JATIT & LLS. All rights reserved, www.jatit.org, TEST CASE PRIORITIZATION TECHNIQUES, SIRIPONG ROONGRUANGSUWAN, 2JIRAPUN DAENGDEJ, Autonomous System Research Laboratory, Science and Technology, Assumption University, Thailand.

[5] Gregg Rothermel, Sebastian Elbaum, Alexey Malishevsky, Praveen Kallakuri and Brian Davia, "The Impact of Test Suite Granularity on the Cost-Effectiveness of Regression Testing", 2001.

[6] Arup Abhinna Acharya, Durga Prasad Mohapatra, And Namita Panda," Model Based Test Case Prioritization for Testing Component Dependency in CBSD Using UMLSequence Diagram", (*IJACSA) International Journal of Advanced Computer Science and Applications,Vol. 1, No. 6, December 2010.*

[7] H. Leung and L. White. Insights into regression testing. In Proceedings of the *International Conference on Software Maintenance*, Miami, Florida, U.S.A., pages 60-69, Oct 1989.

[8] Jefferson Offutt, Jie Pan and Jeffery M. Voas, "Procedures for Reducing the Size of Coveragebased Test Sets", 1995.

[9] Dennis Jeffrey and Neelam Gupta, "Test Case Prioritization Using Relevant Slices", In *Proceedings of the 30th Annual International Computer Software and Applications Conference*, Volume 01, 2006, pages 411-420, 2006.

[10] David Leon and Andy Podgurski, "A Comparison of Coverage-Based and Distribution-Based Techniques for Filtering and Prioritizing Test Cases", *Proc. Int'l Symp. Software Reliability Eng.*, pp. 442-453, 2003.

[11] Cem Kaner, "Exploratory Testing", Florida Institute of Technology, Quality Assurance Institute Worldwide Annual Software Testing Conference, Orlando, FL, 2006.

[12] K. R. Walcott, M. L. Soffa, G. M. Kapfhammer and R. S. Roos, "Time-Aware Test Suite Prioritization", In *Proceedings of the International Symposium on Software testing and Analysis*, pages 1-12, 2006.

[13] Jung-Min Kim, Adam Porter and Gregg Rothermel, "An Empirical Study of Regression Test Application Frequency", ICSE2000, 2000.

[14] Gregg Rothermel, Mary Jean Harrold, Jeffery von Ronne and Christie Hong, "Empirical Studies of Test-Suite Reduction", In Journal of Software Testing, Verification, and Reliability, Vol. 12, No.4, 2002.

[15] John Joseph Chilenski and Steven P. Miller, "Applicability of Modified Condition/Decision Coverage to Software Testing", Software Engineering Journal, Vol. 9, No. 5, pp.193-200, 1994.

[16] H. Do, G. Rothermel, and A. Kinneer, "Prioritizing JUnit Test Cases: An Empirical Assessment and Cost-Benefits Analysis," Empirical Software Eng.: An Int'l J., vol. 11, no.1, pp. 33-70, March 2006.

[17] H. K. N. Leung and L. White, "A cost model to compare regression test strategies", In *Proc. Conf. Softw. Maint.*, pages 201–208, 1991.

[18] S. Elbaum, D. Gable, and G. Rothermel. Understanding and measuring the sources of variation in the priori- tization of regression test suites. In *Proceedings of the Seventh International Software Metrics Symposium*. Institute of Electrical and Electronics Engineers, Inc., April 2001a.

[19] Hyunsook Do and Gregg Rothermel, "On the Use of Mutation Faults in Empirical Assessments of Test Case Prioritization Techniques", IEEE Transactions on Software Engineering, V. 32, No. 9, pages 733- 752, 2006.

[20] G. Rothermel, R. Untch, C. Chu, and M. Harrold. Test case prioritization: An empirical study. In *Proceedings of the International Conference on Software Maintenance*, pages 179{188, 1999.

[21] G. Rothermel, R. Untch, C. Chu, and M. Harrold. Test case prioritization. *IEEE Transactions on Software Engineering*, 27(10):929{948, October 2001.

[22] S. Elbaum, A. Malishevsky, and G. Rothermel. Incorporating varying test costs and fault severities into test case prioritization. In *International Conference on Software Engineering*, pages 329{338, May 2001b.

[23] S. Elbaum, A. Malishevsky, and G. Rothermel. Test case prioritization: A family of empirical studies. *IEEE Transactions of Software Engineering*, 28(2):159{182, February 2002.

[24] G.Rothermel, R. H. Untch, C. Chu and M. J. Harrold. *Prioritizing test cases for regression testing. IEEE Transactions of Software Engineering*, 27(10): 929–948, 2001.

[25] J.-M. Kim and A. Porter. A history-based test prioritization technique for regression testing in resource constrained environments. In *Proceedings of the International Conference on Software Engineering*, May 2002

[26]G. Rothermel and M. Harrold. Analyzing regression test selection techniques. *IEEE Transactions on Software Engineering*, 22(8):529{551, Aug. 1996.

[27] David Leon and Andy Podgurski, "A Comparison of Coverage-Based and Distribution-Based Techniques for Filtering and Prioritizing Test Cases", *Proc. Int'l Symp. Software Reliability Eng.*, pp. 442-453, 2003.

[28] B. Korel and J. Laski, "Algorithmic software fault localization", Annual Hawaii International Conference on System Sciences, pages 246–252, 1991.

[29] Dennis Jeffrey and Neelam Gupta, "Test Case Prioritization Using Relevant Slices", In *Proceedings of the 30th Annual International Computer Software and Applications Conference*, Volume 01, 2006, pages 411-420, 2006.

[30] Journal of Theoretical and Applied Information Technology, © 2005 - 2010 JATIT & LLS. All rights reserved, www.jatit.org, TEST CASE PRIORITIZATION TECHNIQUES, SIRIPONG ROONGRUANGSUWAN, 2JIRAPUN DAENGDEJ, Autonomous System Research Laboratory, Science and Technology, Assumption University, Thailand.

[31] A. G. Malishevsky, J. Ruthruff, G. Rothermel, and S. Elbaum, Cost-cognizant Test Case Prioritization, *Technical Report TR-UNL-CSE-2006-0004,* Department of Computer Science and Engineering, University of Nebraska - Lincoln, March, 2006.

[32] Z.Li, M. Harman and R.M. Hierons. Search Algorithms for Regression Test Case Prioritization.*IEEE Transactions on Software Engineering*, 33(4):225-237, April 2007.

[33] S. Elbaum, G. Rothermel, S. Kanduri and A. G. Malishevsky. Selecting a Cost-Effective Test Case Prioritization Technique. *Software Quality Journal*, 12(3): 185-210, 2004

[34] Gregg Rothermel and Mary Jean Harrold, "A Framework for Evaluating Regression Test Selection Techniques", 1994.

[35] Hema Srikanth, Laurie Williams and Jason Osborne, "System Test Case Prioritization of New and Regression Test Cases", In *Proceedings of the 4th International Symposium on Empirical Software Engineering (ISESE)*, pages 62–71. IEEE Computer Society, 2005.

[36] Jefferson Offutt, Jie Pan and Jeffery M. Voas, "Procedures for Reducing the Size of Coverage based Test Sets", 1995.

[37] Erwan Brottier, Franck Fleurey, Jim Steel, Benoit Baudry and Yves Le Traon, "Metamodel-based Test Generation for Model Transformations: an Algorithm and a Tool", 17th International Symposium on Software Reliability Engineering (ISSRE'06), 2006.

[38] B. Korel and J. Laski, "Algorithmic software fault localization", Annual Hawaii International Conference on System Sciences, pages 246–252, 1991.

[39] Cem Kaner, "Exploratory Testing", Florida Institute of Technology, Quality Assurance Institute Worldwide Annual Software Testing Conference, Orlando, FL, 2006.

[40] David Leon and Andy Podgurski, "A Comparison of Coverage-Based and Distribution-Based Techniques for Filtering and Prioritizing Test Cases", *Proc. Int'l Symp. Software Reliability Eng.*, pp. 442-453, 2003.

[41] D. Binkley, "Using semantic differencing to reduce the cost of regression testing", In the Intl. Conf. On Software Maintenance, pages 41–50, 1992.

[42] Gregg Rothermel, Sebastian Elbaum, Alexey Malishevsky, Praveen Kallakuri and Brian Davia, "The Impact of Test Suite Granularity on the Cost-Effectiveness of Regression Testing", 2001.

AUTHORS

First Author – Rimmi Saini, Greater Noida Group of Institutions, rimmi_saini@yahoo.com

Second Author – Satinder Saini, Assistant Professor, Lovely Campus, Lovely Professional University, Phagwara, Jalandhar, Punjab, satinder_saini@hotmail.com

Third Author – Deepa Gupta, Assistant Professor, Amity Institute of Information Technology, Amity University Campus, Sec-125, Noida, dgupta@amity.edu

Fourth Author –Dr (Prof.) Ajay Rana, Amity School of Engineering and Technology, Amity University, Sec-125, Noida, ajay_rana@amity.edu