Scientific Journal of Impact Factor (SJIF):3.134

e-ISSN (O): 2348-4470
p-ISSN (P): 2348-6406

International Journal of Advance Engineering and Research Development

Volume 2, Issue 8, August -2015

AN OVERVIEW OF TEST CASE PRIORITIZATION TECHNIQUES

D. Saravana Prakash[1], **DR. T. SANTHA**[2]

[1]*Ph.D (PT) Research Scholar,*
School of Information Technology & Science,
Dr.GRD College of Science and Commerce, Coimbatore

[2]*Principal,*
Dr.GRD College of Science and Commerce, Coimbatore

Abstract :- *Test case prioritization techniques engage preparation over test cases in an order that improves the performance of regression testing. It is inefficient to re-execute every test cases for every program function if once change occurs. Test case prioritization is to be planned based on higher priority than lower priority to meet some performance goal. The crisis of test case selection can be solved by prioritizing the test case. The key aim of this paper is to determine the effectiveness of prioritized and nonprioritized test case with the help of APFD(Average Percentage Faults Detected). Test case prioritization techniques could be of great benefit to increasing the effectiveness of test suites in practice. Test case prioritization is a technique helps to increase the rate of fault detection. In this research work, various test case prioritization techniques are analyzed with regression techniques and Genetic algorithm.*

Keywords: *Test Case Prioritization, Regression Testing, Average Percentage of Faults Detected (APFD), Test Cases.*

I.INTRODUCTION

Performance is a essential quality metrics for software system. It can directly affect user understanding and work efficiency. For example, a 500 ms latency increase could cause 20% traffic loss for Google. As another example, the Colorado Benefits Management System is designed to make social welfare accessible. But it runs so slowly that the system is virtually unable to accept assistance applications.
Regression means retesting the unaffected parts of the application. Test cases are re-executed in order to check whether earlier functionality of application is working fine and new changes have not introduced any new bugs. This test can be performed on a new build when there is significant change in original functionality or even a single bug fix. This is the method of verification. Verifying the bugs are fixed and the newly added features have not created any problem in previous working description of software. Testers perform functional testing when new build is available for verification. The aim of this test is to verify the changes made in the existing functionality and newly added functionality. When this test is done , the tester should verify if the existing functionality is working as expected and new changes have not introduced any defect in functionality that was working before this change. Regression test should be the part of release cycle and must be considered in test estimation.

II.RELATED WORK

Ledru et al. [1] used string distances for test case prioritization, incorporating Hamming, Levenshtein, Cartesian, and Manhattan distances. They empirically evaluated the effects of string edit distances on test case prioritization. All test cases were viewed as strings. Before executing test cases, test cases were ordered based on the similarity of strings. In that sense our study is very similar to their study; that is, both of them evaluated the effects of different distance measures on test case prioritization. The most distinctive difference is that their method is applied to black-box testing; that is, test case prioritization is performed before executing the test cases. On the contrary, test case prioritization in our study is performed after executing the test cases. Moreover, the distance measures we applied are different from theirs. Similarly, Hemmati et al.

introduced a family of similarity-based test case selection techniques for model-based testing [2]. They also analyzed the impact of similarity functions on similarity-based test case selection [3]. In model-based testing, test cases generated from UML state machines are abstract rather than concrete. The method proposed by Hemmati et al. is applied to select a subset of the generated test suite without considering the order of selected test cases. Unlike the above two methods, our approach uses dynamic profile information generated by executing test cases to reschedule the execution order of test cases.

Jiang et al. [4] proposed a new family of coverage-based ART techniques, which were statistically superior to the RT-based techniques in detecting faults. Fang et al. [5] introduced several similarity-based test case prioritization techniques based on the edit distances of ordered sequences. The execution profiles of test cases were transformed into the ordered sequences of program entities (e.g., statement or branch). The distance of two test cases was defined as the edit distance of their ordered sequences calculated by the proportional distance.

Korel et al. [7] implemented an experiment in order to validate and verify efficiency and effectiveness of both simple code based and model based test case prioritization. The Aim of this experiment was to verify these approaches to evaluate the performance of earlier fault detection in the system that has modified. So the result shown that as the execution of model based test cases are very quick as compared to code based test case prioritization so it can bring improvement in earlier detection of fault. That's why for the whole test suites the model based test prioritization in comparison of code based test case prioritization is cheap.

Korel et al. [8] in their paper prioritized the test cases by applying certain model based test prioritization heuristics. It has some problems that selective model based prioritization focus only the number of identified transitions which does not have valuable impact on the improvement of earlier detection of faults. Value based regression test case prioritization is used for the earlier fault detection. For prioritization of test cases, they proposed an algorithm on the basis of 6 factors. Modifications in requirement, Priority of customer, complexity in Implementation, traceability in requirement, time of execution and Impact of fault. The PSO (Particle swarm Optimization) is applied for allocating utility to factors and comparison of factor's values. Maximum value is the highest number, and minimum value is the lowest number. For all the test cases, sum up the values of factors. If the factor values of two test cases match, then it shows by comparing requirement utilities of those test cases the judgement is made. This implies, in terms of time and cost PSO algorithm is much efficient, powerful and useful than greedy algorithm.

Parakash et al [9] invented a new method for test case prioritization known as "potentially weighted method". This approach prioritizes test cases based on "potential coverage" like coverage of code, function, branch, fault, and path and for criteria weights are assigned. Test cases are given preference on the basis of value of weight, and the weight is from high to low. Every statement in code must be executed at least once; this is the basic purpose of code coverage testing.

Kavitha et al [10] invented a technique on the basis of rate of fault impact and fault detection to prioritize the test cases. For identification of dangerous faults at earlier unique algorithm is invented. The invented algorithm determines the test case weight age.

$$Tcw = RFT + Fl$$

Tcw denotes test case weightage age.

$$Tcw = RFTi + Fli$$

RFTi denotes fault detection rate, average no of faults per minute by test case, is knows as fault detection.

Using genetic algorithm [11] invented testing that includes determination of the test cases, which can able to detect bugs in system. The process is tough and time taking. Author proposed a new technique in this paper, for test case prioritization according to their capabilities of discovering bugs. Higher priority is assigned to more similar errors. Low priority is assigned to less similar errors. Through genetic algorithm this order will be achieved. For finding the fittest chromosome like selection, crossover, mutation is applied on chromosomes. Genetic algorithm Steps includes: Set population, find fitness of population, for individual apply selection, apply crossover and mutation, figure out and recreate chromosome. Approximately an optimized solution for large number of time will be provided.

Amitabh et al [12] invented a prioritization technique on the basis of binary code. They delivered a system called ECHELON. On the basis of modifications that are being done in the program it prioritizes the test cases. ECHELON is a unified part of Microsoft development process. It uses simple and quick algorithm. It give results within few seconds by saving time and resources.

H.Do et al [13] present a controlled experiment. Software developers used Junit framework for generating test cases that are being executed in java. Junit provides helps to testers to create test cases and to re execute these test cases when modifications occur in program. There experiment is for finding the efficiency and effectiveness of test case prioritization under this JUnit Framework. They developed 6 blocks and method level techniques which are as follows. 1) Total block coverage 2) Additional block coverage 3) Total method coverage 4) Additional method coverage 5) Total DIFF method 6) Additional DIFF method.

III. TECHNIQUES

Different Prioritization Techniques

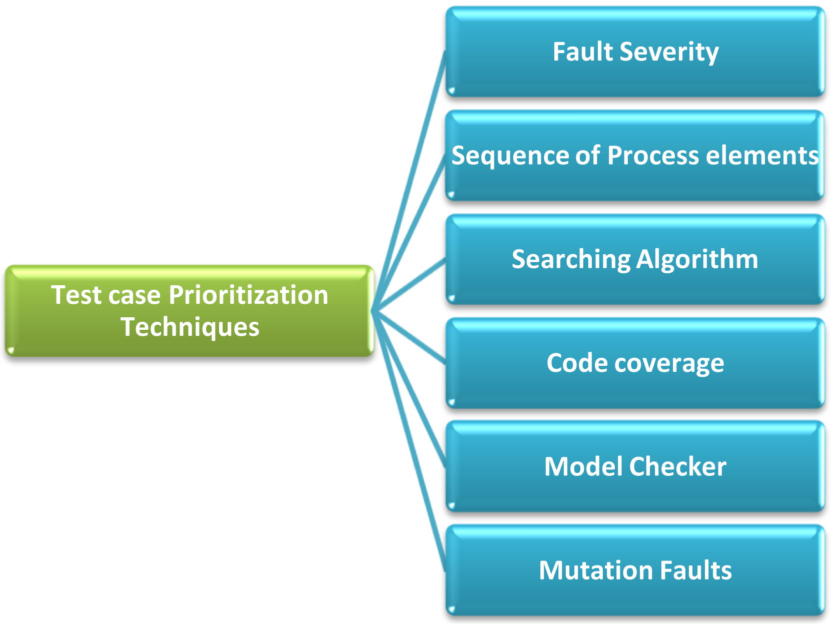


Fig 1: Prioritization of Techniques

COVERAGE-BASED PRIORITIZATION TECHNIQUES

Coverage-based prioritization techniques have been widely studied. These techniques can usually be implemented at three different coverage levels: function, statement, and branch.

(i) Total Coverage Maximization Prioritization. Total coverage maximization prioritization techniques schedule test cases in descending order of total coverage achieved. When more than one test case has the same total coverage, the algorithm selects one at random. The prioritization process is repeated until all test cases have been prioritized.

(ii) Additional Coverage Maximization Prioritization. Additional coverage maximization prioritization techniques work as follows: A test case that yields the greatest coverage is selected at first. And then, the next test case to be selected is the one that can cover the maximum program entities not yet covered by previously selected test cases. The process is repeated until all program entities have been covered by at least one test case.

Having prioritized test cases in this manner, if remaining test cases cannot add the additional coverage, we schedule them in the same way. If more than one test case yields the same additional coverage in the process of prioritization, we randomly select one. Additional coverage maximization prioritization is based on feedback. In other words, previously selected test cases are used to provide guidance to select next test case. On the contrary, total coverage maximization prioritization is no feedback. For a test suite and program with m test cases and n statements, respectively, total coverage maximization prioritization requires time $O(mn)$, while the cost of additional coverage maximization prioritization is $O(m^2n)$.

GLOBAL SIMILARITY-BASED PRIORITIZATION ALGORITHM

The diversity of test cases aids in improving their fault revealing power [6]. ART-based prioritization algorithm selects a test case that is the farthest from the prioritized test suite from a candidate set. Since the candidate set is a subset of not yet prioritized test cases, this algorithm does not guarantee the diversity of test cases to a maximum extent. Inspired by Hemmati et al. we propose a global similarity-based test case prioritization algorithm. This algorithm selects a test case from all not yet prioritized test cases, rather than a subset of them. Consequently, it can maximize the diversity of prioritized test cases.

COUPLING IN OBJECT-ORIENTED PROGRAMMING

Coupling is defined as the degree of interdependence between two modules. However, in an object-oriented programming environment, coupling can exist not only at the level of methods but also at the class level and package level. Therefore, coupling represents the degree of interdependence between methods, between classes, between packages, and so forth. Many researchers have proposed different slicing based mechanisms to measure coupling in an object-oriented framework. There are many factors, such as information hiding, encapsulation, inheritance, message passing, and abstraction mechanisms, that contribute to coupling in object-oriented programs. High coupling affects program comprehension and analysis. As a result, it becomes very difficult to maintain software systems. In an object-oriented program, coupling can exist between any two components due to message passing, polymorphism, and inheritance mechanisms of object-oriented programs. These components include packages, classes, methods, and statements. Two statements s1 and s2 are said to be coupled if s1 has some dependence (control, data, or type dependence) on s2 . Methods in an object-oriented program belong to the constituent classes. It implies that a method is coupled either with a method in the same class or with another method in a different class. If the methods of any two classes are coupled, then the corresponding classes are said to be coupled. Similarly, the container packages of the coupled classes are also said to be coupled.

REGRESSION TESTING

A software regression means the loss of some features in a new version of software that usually results in an error message, a logical error or the lack of action. Regressions are caused by changing some parts of the program code. As a consequence of these changes some functions may stop working. To find such errors, the regression testing may be used. It is a type of software testing able to determine the correctness of functions that were present in previous versions of a program. Regression testing reveals defects caused accidentally during the process of program optimization. Specifying which test cases should be performed first in regression testing, can significantly increase the likelihood that a potential user will get a working stable version of software in relatively short time.

REGRESSION TESTING TECHNIQUES

There are number of available regression testing techniques. Here we are representing all these techniques in basic 3 categories as defined .

(i) Retest All: As the name suggest in this testing technique we perform whole testing cycle again after the inclusion of new code and component and related test cases into it. Again the test cases will be generated, sequence reset etc. This type of technique is not feasible in most of time, as it requires much time and cost. But in smaller software where a small change in code impact on whole software at that time regression testing is used.

(ii) Regression Test Selection: This approach is a modification over the existing retest all approaches. In this approach instead of testing all cases a selection on the test cases is performed. To perform this selection a test cases categorization is performed. According to this rest table cases are separated from whole test cases such as the requirement based testing is generally need not to be performed again. The code based test cases and the system based test cases are selected to perform the testing process. In this technique instead of rerunning the whole test suite, we select a part of test suite to rerun if the cost of selecting a part of test suite is less than the cost of running the tests that RTS allows us to omit. RTS divides the existing test suite into (1) Reusable test cases; (2) Re-testable test cases; (3) Obsolete test cases.

(iii) Test Case Prioritization: All the test cases used in a testing approach or the sequence are not alike. It means each kind of test cases have there on values called the basic prioritization of the test cases. Generally the prioritization process is defined on the bases of state space diagram of the cases. The test cases that exist on initial stage of the test cases or the development process have the lower priority and the test cases that affect the whole system or tested repeatedly over the whole process having the higher priority. Besides this the prioritization process is further divided in number of sub techniques to assign the priorities a)The easiest type of assigning priorities is the random prioritization but in most of the cases it does perform the complete justification with the test cases selection. Because of this such type of technique is never recommended to generate the test cases. (b) Optimal ordering: in which the test cases are prioritized to optimize rate of fault detection. As faults are determined by respective test cases and we have programs with known faults, so test cases can be prioritized optimally. It is one of the dynamic prioritization approach in which decision is affected because of types of occurred faults and there frequency. (c) Total statement coverage prioritization: in which test cases are prioritized in terms of total number of statements by sorting them in order of coverage achieved. If test cases are having same number of statements they can be ordered pseudo randomly.

GENETIC ALGORITHM (GA)

A genetic algorithm (GA) is an optimization technique [22], solicited to different real time problems.GA is a method for solving both constrained and unconstrained optimization problems that is based on natural selection, the process that drives biological evolution. GA repeatedly modifies a population of individual solutions. At each step, the genetic algorithm selects individuals at random from the current population to be parents and uses them produce the children for the next generation. Over successive generations, the population "evolves" toward an optimal solution. GA can also be applied in NP-hard problems.

ANT COLONY OPTIMIZATION (ACO)

Swarm intelligence is a known approach to problem solving which extracts inspiration from nature biological systems. Ant colony optimization (ACO), introduced by Dorigo in his doctoral dissertation, is a class of optimization algorithms modeled on the actions of an ant colony. ACO is a probabilistic technique useful in problems that deal with finding better paths through graphs. Artificial 'ants'—simulation agents—locate optimal solutions by moving through a parameter space representing all possible solutions. Natural ants lay down pheromones directing each other to resources while exploring their environment. The simulated 'ants' similarly record their positions and the quality of their solutions, so that in later simulation iterations more ants locate better solutions.

ARTIFICIAL BEE COLONY OPTIMIZATION (ABC)

Artificial Bee Colony (ABC) algorithm is a swarm based meta-heuristic algorithm. The colony of bees in ABC algorithm consists of three groups of bees: employed bees, onlookers and scouts . Employed bees forage in search of their food source and return to hive and perform a dance on this area. The employed bee who find abandoned food source becomes a scout and find a new food source again. Onlookers decided their food source depending upon the dances of employed bees. A nectar source is chosen by each bee by succeeding a nest mate whose food source has already discovered. The bees dance on the hive, to inform that they discovered of nectar sources and persuade their nest mates to follow them. To get nectar, other bees follow the dancing bees to one of the nectar areas. On collecting the nectar they come back to their hive, handover the nectar to a food storer bee. After renounce the food, the bee opts for one of the choices with a certain probability (a) Leave the food source and act as a nonaligned follower, (b) Without joining the nest mates, continue to forage at the food source or (c) volunteer the nest mates by dancing before the return to the food source. Various food areas are identified and announced by bee dancers within the dance area. The policy, by which the bee decides to follow a specific dancer, is not revealed yet but it is taken as “the recruitment among bees is always a function of the quality of the food source”.

MULTI-OBJECTIVE GENETIC ALGORITHM

GA is a population based approach and is well suited for solving multi-objective optimization problems. In a single run a generic singleobjective GA can be easily modified to find a set of multiple non-dominated solutions. The capability of GA to simultaneously search different regions of a solution space makes it possible to find a diverse set of solution for difficult problems with non-convex, discontinuous, and multi-model solutions spaces. Many engineering problems have multi=objectives, including engineering system design and reliability optimization. The available common techniques used in Multi-objective GA to attain the goals of multi-objective optimization are as below: 1. Fitness functions 2. Diversity 3. Elitism 4. Constrain Handling 5. Parallel and Hybrid Multi-Objective GA.

PRIORITIZED TEST SUITE EFFECTIVENESS

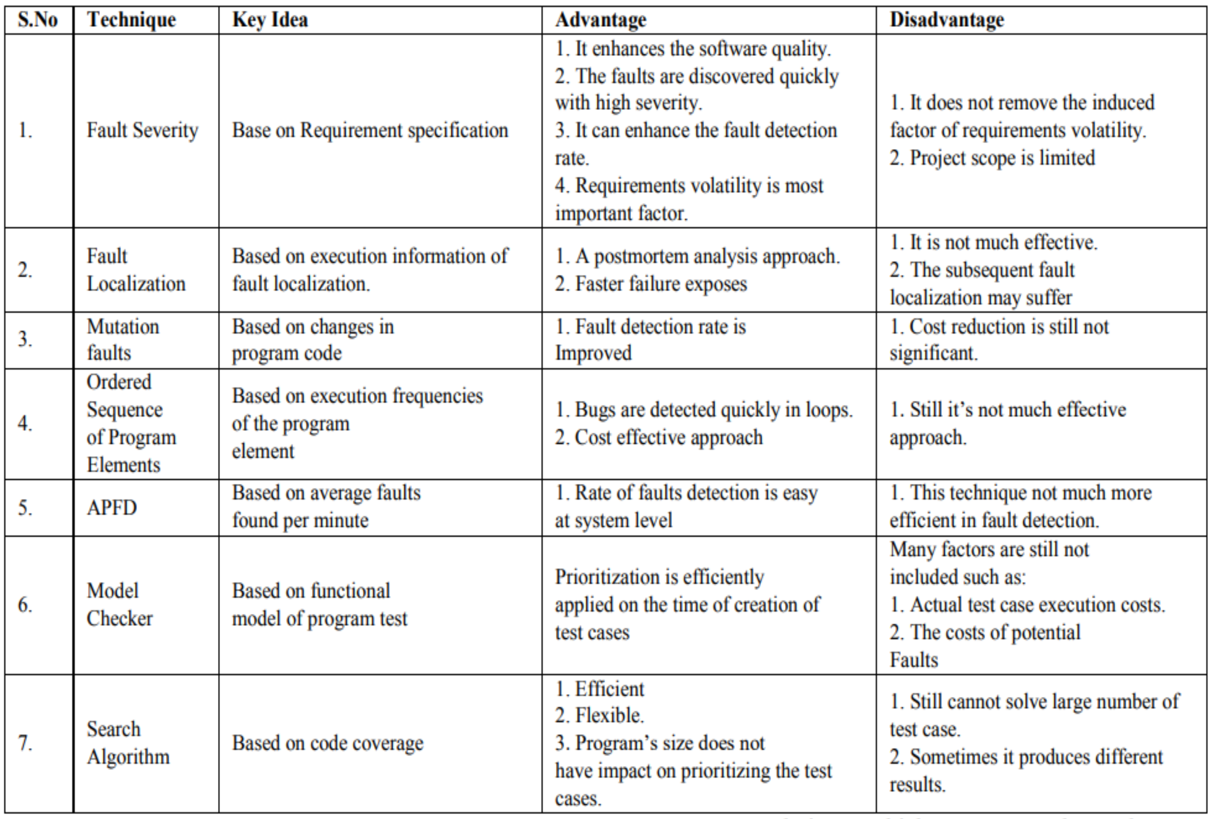
The performance of the prioritization technique used in this paper, it is must to assess effectiveness of the sequence/ordering of the test suite. Effectiveness will be measured by the rate of faults detected. The following metric is used to calculate the level of effectiveness. 3.1 Average Percentage Of Faults Detected (APFD) Metric To quantify the goal of increasing a subset of the test suite's rate of fault detection, we use a metric called APFD developed by Elbaum et al.[6] that measures the rate of fault detection per percentage of test suite execution. The APFD is calculated by taking the weighted average of the percentage of faults detected during the execution of the test suite. APFD values range from 0 to 100; higher values imply faster (better) fault detection rates.

APFD can be calculated as follows:

$$APFD=1-\{(Tf1+Tf2+\ldots+Tfm)/mn\}+(1/2n) \ldots\ldots\ldots\ldots\ldots\ldots (1)$$

Where n be the no. of test cases and m be the no. of faults. (Tf1,….,Tfm) are the position of first test T that exposes the fault. Here comparison among the results of prioritized and nonprioritized suite is done based on the results of the APFD metric. This is average percentage of faults detected. APFD is a standardized metric that is used to find the degree of faults detected.
The prioritized order according to fi is:
T4 T2 T1 T7 T6 T9 T10 T5 T8 T3
No. of test cases (n) = 10
No. of faults (m) = 10
The position of the first test in T that exposes fault i. = Tfi
Applying APFD w.r.t. the prioritized test cases:
APFD = 1 – {(5 + 2 + 4 + 1 + 2 + 3 + 3 + 4 + 1 + 2) / (10*10)} + {1/(2*10)}
= 1 – { 27 / 100} + { 1 / 20}
= 1 – 0.27 + 0.0
= 0.78
Now APFD value for non – prioritized test cases:
APFD = 1 – {(6 + 2 + 7 + 4 + 2 + 1 + 1 + 7 + 4 + 2) / (10*10)} + {1/(2*10)}
= 1 – { 36 / 100} + { 1 / 20}
= 1 – 0.36 + 0.05
= 0.69

Comparison of Test case Prioritization Techniques



S.No	Technique	Key Idea	Advantage	Disadvantage
1.	Fault Severity	Base on Requirement specification	1. It enhances the software quality. 2. The faults are discovered quickly with high severity. 3. It can enhance the fault detection rate. 4. Requirements volatility is most important factor.	1. It does not remove the induced factor of requirements volatility. 2. Project scope is limited
2.	Fault Localization	Based on execution information of fault localization.	1. A postmortem analysis approach. 2. Faster failure exposes	1. It is not much effective. 2. The subsequent fault localization may suffer
3.	Mutation faults	Based on changes in program code	1. Fault detection rate is Improved	1. Cost reduction is still not significant.
4.	Ordered Sequence of Program Elements	Based on execution frequencies of the program element	1. Bugs are detected quickly in loops. 2. Cost effective approach	1. Still it's not much effective approach.
5.	APFD	Based on average faults found per minute	1. Rate of faults detection is easy at system level	1. This technique not much more efficient in fault detection.
6.	Model Checker	Based on functional model of program test	Prioritization is efficiently applied on the time of creation of test cases	Many factors are still not included such as: 1. Actual test case execution costs. 2. The costs of potential Faults
7.	Search Algorithm	Based on code coverage	1. Efficient 2. Flexible. 3. Program's size does not have impact on prioritizing the test cases.	1. Still cannot solve large number of test case. 2. Sometimes it produces different results.

Table 1: Comparison of Test Case Prioritization Techniques

IV.CONCLUSION

A benchmark study was conducted on few of the most accepted optimization techniques. Software testing is one of the cost consuming activity but mandatory for quality assurance in software development lifecycle. By reducing the number of test cases or test suites the software testing cost can be considerably reduced. This research paper analyzes the test case optimization techniques in the field including probabilistic, meta-heuristic, Multi-objective optimization.

V.REFERENCES

1.Y. Ledru, A. Petrenko, and S. Borody, “Prioritizing test cases with string distances,” Automation Software Engineering, vol. 19, no. 1, pp. 5–95, 2012.
2.H. Hemmati, A. Arcuri, and L. Briand, “Achieving scalable model-based testing through test case diversity,” ACM Transactions on Software Engineering and Methodology, vol. 22, no. 1, article 6, 2013.
3.H. Hemmati and L. Briand, “An industrial investigation of similarity measures for model-based test case selection,” in Proceedings of the IEEE 21st International Symposium on Software Reliability Engineering (ISSRE '10), pp. 141–150, San Jose, Calif, USA, November 2010.
4.B. Jiang, Z. Y. Zhang, W. K. Chan, and T. H. Tse, “Adaptive random test case prioritization,” in Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering (ASE '09), pp. 233–244, Auckland, New Zealand, November 2009.
5.C. Fang, Z. Chen, K. Wu, and Z. Zhao, “Similarity-based test case prioritization using ordered sequences of program entities,” Software Quality Journal, vol. 22, no. 2, pp. 335–361, 2014.
6.H. Hemmati, A. Arcuri, and L. Briand, “Achieving scalable model-based testing through test case diversity,” ACM Transactions on Software Engineering and Methodology, vol. 22, no. 1, article 6, 2013.
7. B. Korel, G. Koutsogiannakis, “Experimental Comparsion of Code Based and Model model Based Test prioritization,” IEEE 2009.
8.B. Korel, G. Koutsogiannakis, and L.H.Tahat, “Application of System Models in Regression Test Suite Prioritization,” in Proceedings of the 24thIEEE International Conference Software Maintenance (ICSM ’08) pp.247- 256, 2008.
9.Prakash.N, Rangaswamy “Potentially weighted method for test case prioritization” JCIS 9:18(2013) 7147-715
10.R.Kavitha, N.Sureshkumar “Test case prioritization for regression testing based on severity of fault”, IJCSE vol.02, no.05 2010, 1462- 1466.
11.Ruchika malhotar, abhishek bharadwaj “Test case prioritization using genetic algorithm” IJCSI: 2231-5292, vol-2, Issue-3, 2012.
12.A.Srivastava, and J.Thiagarajan, “Effectively prioritizing tests in development environment”, Proceedings of the International Symposium on Software Testing and Analysis, pp.97-106, July 2002.
13.H.Do, G.Rothermel and Kinner, “Empirical studies of test case prioritization in a Junit testing environment”, Proceeding of the International Symposium on software Reliability Engineering, pp.113- 114, NOV 2004.