Novel Classification of Test Case Prioritization Techniques

Kamna Solanki
Assistant Professor
University Institute of Engg and Technology
M.D University, Rohtak, India

Yudhvir Singh, Ph.D
Associate Professor
University Institute of Engg and Technology
M.D University, Rohtak, India

ABSTRACT
Test case prioritization techniques schedule test cases to reduce the cost of regression testing and to maximize some objective function. Test cases are prioritized such that those test cases which are more important under some criteria are executed earlier in regression testing process. The various objective functions are applicable as a metric of how rapidly faults are discovered during the testing process like rate of fault detection. Therefore, prioritization techniques are effective when implemented for specific instances. In this paper, a novel classification for test case prioritization is made which may cover every concept or measure and contribute for improvement of regression testing process.

General Terms
Regression testing: Test suites are saved so that they can be reused after the evolution of the software. This reuse of test suite is called the regression testing.

Keywords
Test case, Test Case Prioritization, test case prioritization techniques

1. INTRODUCTION
The evolution of computer-based systems and products in the current scenario of globalization is derived from software which is one of the most important technologies and has grown from being a mere problem solving tool, a lot is yet to be done on the development of quality software that performs the right job at the right time. It is here that software engineering intends to provide a structured framework for building high quality software **[1]**.It is with (SDLC) is framed, which is a series of steps that are to be followed in an order to produce efficient software which is cheaper. Amongst the different steps in SDLC, software testing is a very important yet a mandatory step, which ensures the proper working of the software.

For evaluating a system or attribute or capability of a program software testing is the capable process and also for determining through the purpose to find that whether it satisfies or meets the specified requirements or not. In simple words testing is executing a system in order to discover any errors gaps or missing requirements in contrary to the actual requirements desire output **[2]**. Software testing consists of following steps: design of test cases, preparing the test data, execution of program with test data, and the last but not the least comparing the result with that of test case. Design or generation of test cases is a challenging part.

Software testing is of different types at different levels. One of the types of testing is regression testing which detects errors after modifications in the present system. Regression testing is a costly testing process used for validation of modified software and detection of new faults introduced into earlier tested code. Regression test suites can be expensive to execute fully; thus, test cases are prioritized, they are assigned a priority by some criteria such that which are more important are executed prior in regression testing process.
There are four methods for regression testing
These methods are: **[3] [4] [5]**
1. Retest all
2. Regression Test Selection
3. Test Suite Reduction
4. Test Case Prioritization

The purpose of this prioritization is to increase the chances that if the test cases are used for regression testing in the given order, they will more strongly meet some objectives than they would if they were executed in some other order.

2. RELATED WORK
Wong et al. suggested prioritizing test cases according to the measures of increasing cost per coverage added. In the testing process, an indirect objective was to reveal faults earlier by the method of ranking. The authors restricted their attention to prioritization of test cases for execution on a specific modified version of a program, and to prioritization of only the subset of test cases selected by a safe regression test selection technique from the test suite for the program. The authors did not specify a mechanism for prioritizing the remaining test cases after full coverage has been achieved. The authors described a case study in which their technique is applied to a program of 5000 lines of code, and evaluated against ten faulty versions of that program, and concluded that the technique was cost-effective in that application **[6]**.

Rothermel et al. and Elbaum et al. provided the first formal definition of the prioritization problem and presented metrics for measuring the rate at which faults are discovered in test suites. The authors defined prioritization techniques, and presented the results of several observed studies of those techniques **[7] [8]**. Jones et al., described a technique for prioritizing test cases for use with the modified condition/decision coverage (MCDC) criteria, this technique uses feedback, but no modification information **[9]**.

Srivastava et al. presented a technique for prioritizing test cases based on basic block coverage, using both feedback and change information. The technique was different from previous techniques as it computes flow graphs and coverage

from binaries, and attempts to predict possible affects on control flow following from code modifications. The authors described the application of this technique to several large systems at Microsoft, and provided data showing that the approach can be applied efficiently to those systems **[10]**.

Avritzer et al. presented techniques for generating test cases which can be applied to software that can be modeled by Markov chains, having a condition that operational profile data is available. Although the authors do not use the term "prioritization", their techniques generate test cases in an order that can cover a larger proportion of the software states; most probably to be reached in the field earlier in testing. Essentially, prioritizing the test cases in an order that increases the likelihood that faults more likely to be encountered in the field will be uncovered earlier in testing. The approach provides an example of the application of prioritization to the initial testing of software when test suites are not yet available **[11]**.

Sampath et al. presented the prioritization of test cases for web applications. The test cases were recorded user sessions from the previous version of the SUT, in their case. Session-based test cases were thought to be appropriate for testing web applications because they tend to reflect the actual usage patterns of real users, by making for realistic test cases. They compared different criteria for prioritization such as the number of HTTP requests per test case, coverage of parameter values, frequency of visits for the pages recorded in sessions and the number of parameter values. The empirical studies showed that prioritized test suites performed better than randomly ordered test suites, but also that there is not a single prioritization criterion that is always best. However, the 2-way parameter-value criterion, the prioritization criterion that orders tests to cover all pair-wise combinations of parameter-values between pages as soon as possible, showed the highest APFD value for 2 out of 3 web applications that were studied **[12]**.
Fraser et al. introduced a model-based prioritization approach. This prioritization technique was based on the concept of property relevance **[13]**. A test case is relevant to a model property if it is theoretically possible for the test case to violate the property. The relevance relation is obtained by the use of a model-checker, which is used as the input to the greedy algorithm. While they showed that property-based prioritization can outperform coverage-based prioritization, they noted that the performance of property-based prioritization is heavily dependent on the quality of the model specification **[14]**.

Kim et al. evaluated several regression test selection techniques and a prioritization technique of their own invention that exploits historical execution data. None of the selection or prioritization techniques considered in the studies are distribution-based **[15]**.

3. TEST CASE PRIORITIZATION

Test case prioritization finds the best ordering of test cases for testing, so that the tester obtains maximum benefit, even if the testing is prematurely halted at some random point. The approach was first mentioned by Wong et al. **[16]** . On the other hand, in that work, it was only applied to test cases that were already selected by a test case selection technique.

Test case prioritization can deal with a wide variety of objectives, including the following:
1. Testers may wish to raise the rate of fault detection that is, the chances of revealing faults prior in a run of regression tests.
2. Testers may wish to raise the rate of detection of high-risk faults, locating those faults, prior during the testing process.
3. Testers want to raise the chances of revealing regression errors associated with particular changes in code, prior during the regression testing process.
4. Testers may wish to increase their code coverage in the system under test more rapidly.
5. Testers want to be assured about the reliability of the system which is to be tested, more rapidly.
According to the observation, and on the basis of the preference of goal, the test case prioritization problem may be difficult for certain goals or objectives.

3.1 Test Case Prioritization Problem

The test case prioritization problem can be defined as:

Given: T, a test suite; PT, the set of permutations of T; f, a function from PT to the real numbers.

Problem: Find T' belongs to PT such that (for all T") (T" belongs to PT) (T" $\neq$ T') [f (T') $\geq$f (T")].

Here, PT represents the set of all possible prioritizations of T and f is function that, applied to any such ordering, yields an award value for that ordering **[7][3]**.

4. PRIORITIZATION TECHNIQUES

Various test case prioritization techniques may be utilized to meet the goal, which is to be achieved. For example, to raise the speed of fault detection in test suites, test cases might be prioritized in terms of the extent to which they execute modules that have tended to fail in the past. Otherwise, test cases in terms of their increasing cost-per-coverage of code, or in terms of their increasing cost-per-coverage of features listed in a requirements specification are prioritized **[8]**. In any case, the motive behind the choice of a prioritization technique is to raise the chances that the prioritized test suite can better meet the goal than some arbitrary ordering of test cases.

This section introduces a new "**3CMDHO**" classification of existing test case prioritization techniques which are: (a) **C**ost Based Techniques, (b**) C**hronological History Based Techniques, (c) **C**ustomer-Requirement Based Techniques, (d)**M**aximize Coverage for Early Fault Detection (MCEFD) (e) **D**istribution Based Techniques, (f) **H**ybrid Approaches, (g)**O**ther Approaches .

Test case prioritization techniques provide a method to plan and run test cases according to some priority so as to provide earlier fault detection. The "3CMDHO" classification is shown in fig1:

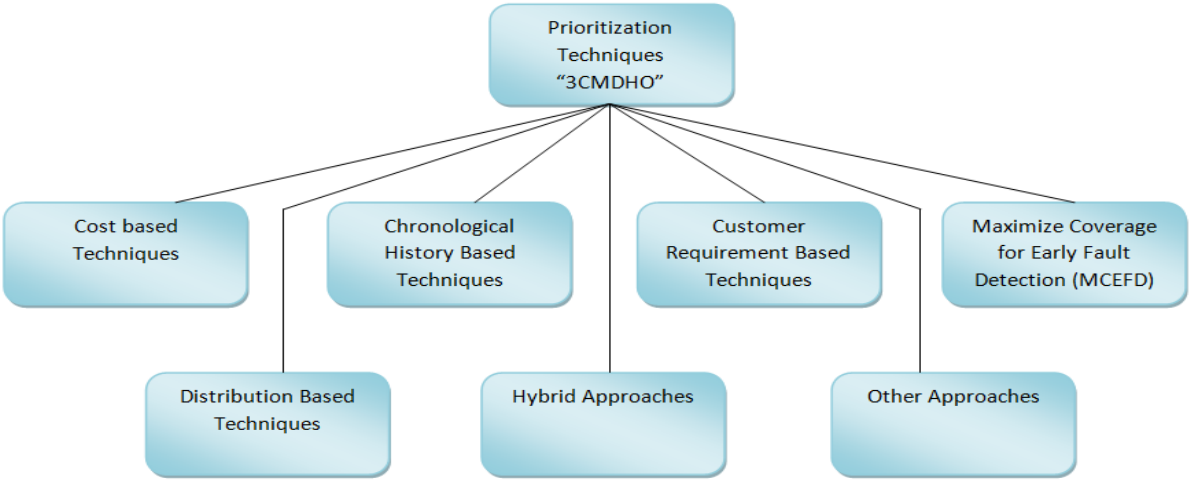

Fig1: Novel Classification of Test Case Prioritization Techniques

4.1 Cost Based Techniques

Cost based techniques are methods of test case prioritization on the basis of costs, like analysis cost and prioritization cost. Many researches are done in this area. The present existing cost based test case prioritization techniques are given next. A cost model for regression test selection is provided by Leung et al. The proposed model includes various costs of regression testing, the execution and validation costs of test cases and the analysis cost to maintain test selection. It provides a way to compare test cases for relative effectiveness. This model can be properly applied to an effective regression test selection techniques, which necessarily select all test cases in the existing test suite that may reveal faults **[17]**. However, in this model the costs of overlooking faults are not considered because of the discarded tests. Malishevsky et al. offered cost models for prioritization which take these costs of overlooking faults into account. The authors used different variables for test case prioritization like : Analysis Cost as Ca(T) ; cost of Prioritization Algorithm as Cp(T).

$$WP = Ca(T) + Cp(T) \quad (7)$$

Where:

• WP – It is weight prioritization value of every test case.

• Ca(T)- It includes source code analysis cost , change analysis between old and new versions and a collection of various execution traces.

• Cp(T)- It is the actual cost which is incurred in running a prioritization tool, based on the prioritization algorithm used.

The authors have categorized the process of regression testing process into two phases named as preliminary phase and critical phase. Activities of Preliminary phase can have variable costs than activities of critical phase, since the later can have more complications for many things such as release time etc. The cost of any particular test case mostly depends on the amount of resources needed to execute and validate that test case. Also, Cost-cognizant prioritization needs an advance idea regarding the brutality of every defect that can be found by a test case. Brutality of any defect can be used for ordering tests using the same criteria **[18]**.

4.2 Chronological History Based Techniques

Chronographic history-based techniques are methods of test case prioritization on the basis of test execution history. Jung-Min et al., proposed a technique which was based on knowledge of prior performance of every test case which decides the chances to include that test case in present testing session. This approach was actually inspired by statistical quality control (exponential weighted moving average) and statistical forecasting (exponential smoothing) **[19]**. Kim et al. described various selection probabilities of each test case, TC, at time, t, to be $P_{tc,t}(H_{tc}, \alpha)$, where H_{tc} is a set of t, time-ordered observations $\{h_1, h_2, \ldots h_n\}$ drawn from runs of TC and α is a smoothing constant used to weight individual historical observations **[15]**. The higher values indicate recent observations, while lower values indicate older values. These values are then normalized and used as probabilities. The general form of:

P is $P_0 = h_1$ and $P_k = \alpha_{hk} + (1- \alpha)P_{k-1}$, $0<=\alpha<=1$, $k>=1$[19].

When testing in a black box environment, source code related information is not available. In those situations, researchers have output of test cases only and other dynamic information, such as the running time of test cases.

4.3 Customer Requirement Based Techniques

Hema et al. offered the requirements-based test case prioritization approach to prioritize a set of test cases. They built upon current test case prioritization techniques and proposed to use several factors to rank the test cases. Those factors are the customer-assigned priority (CP), requirements complexity (RC) and requirements volatility (RV). Additionally, the authors have assigned values (1 to 10) for each factor for the measurement. They declared that higher factor values indicate a need for prioritization of test case related to that requirement. Weight (rank) prioritization (WP) measures the important of testing a requirement earlier **[20]**.

$$WP = \Sigma (PF_{value} * PF_{weight}); PF=1 \text{ to } n \quad (1)$$

Where:

WP denotes weight prioritization that measures the significance of testing a requirement.

• PF_{value} is the value of each factor, like CP, RC and RV.

• PF_{weight} is the weight of each factor, like CP, RC and RV.

Test cases are then ordered such that the test cases for requirements with high WP are executed earlier to others. Recent research showed that using different test case

prioritization techniques can significantly affect the rate of fault detection of the test suite. The tester can choose to arrange the test cases in descending order of their priority values (with arbitrary ordering in case of ties).Hema **[20]** were interested in two particular objectives of test case prioritization approaches: (a) to widen user professed software quality in a cost effective way by considering potential defect severity and (b) for improvement of the rate of discovering harmful faults during system level testing of newly generated code and regression testing of existing modified code. There is a simple approach for test case prioritization which was proposed earlier through the requirement traceability matrix. The matrix can be produced by mapping from use cases in the Use Case diagram to functional requirements from users.

4.4 Maximize Coverage for Early Fault Detection (MCEFD)

This approach combines coverage based and fault based techniques. Structural coverage is a metric that is often used as the prioritization criterion. The intuition behind the idea is that quick maximization of structural coverage increases the probability of quick maximization of fault detection. Therefore, while the goal of test case prioritization is to achieve a higher fault detection rate, prioritization techniques actually focus on maximizing early coverage.

In this technique, focus is on the objective listed of increasing the chances of revealing faults earlier by increasing the coverage in the testing process. The motivation for meeting this objective is clear: faster feedback on the system under test can be provided by regression testing, if rate of fault detection has improved, or earlier proof that quality measures have not been met according to the goals set; it can also let debuggers begin their work earlier. Coverage means code coverage, or structural testing. Structural testing compares test program behavior against the obvious purpose of the source code. This contrasts with functional testing, which compares test program behavior with a requirements specification. It examines how the program works, considering possible pitfalls in the structure and logic.

4.4.1 Comparator Techniques

Random ordering: One of the prioritization techniques is the random ordering of the test cases in the test suite. Random ordering means test cases are arbitrarily arranged in an order.

Optimal ordering: An optimal ordering of the test cases in the test suite can be obtained if faults are known and it can be determined that which faults each test case exposes: this ordering of test cases maximizes rate of fault detection in a test suite. As observed, this is not a practical technique, but it provides an upper bound on the effectiveness of the other heuristics that we consider.

4.4.2 Statement level techniques

Total statement coverage prioritization

While dealing with a program it can be determined easily that which statement can be verified by which test case. Thus these test cases can be prioritized according to the total number of statements it can cover by sorting them in order of total statement coverage achieved.

Additional statement coverage prioritization

Total statement coverage prioritization sorts test cases in the order of total statement coverage achieved. Even after, executing a test case and covering number of statements, many more statements can be obtained which are not covered till now by further testing. Additional statement coverage prioritization chooses a test case which provides the highest statement coverage, after that adjusts the coverage data about subsequent test cases to show their coverage of statements which are not yet covered, and then process is repeated so as all statements are at least covered once by any test case. When all statements are covered, remaining test cases must also be scheduled; we do this recursively by resetting all statements to "not covered" and applying additional statement coverage on the remaining test cases again.

Total FEP prioritization

The ability of a fault to be uncovered by a particular test case actually not only relies on the ability of a test case to execute a defective component, but it also relies on the chance that a defect in that statement will trigger a failure for that particular test case **[21]**. However, practical measurement of this probability must be an approximation; it should be known whether the use of such an approximation might yield a prioritization technique better than any other techniques based only on code coverage, in terms of rate of fault detection than techniques. To approximate the fault-exposing-potential (FEP) of a test case we used mutation analysis **[22]**. Given program P and test suite T, for each test case t 2 T, for each statement s in P, we determined the mutation score ms(s; t) of t on s to be the ratio of mutants of s exposed by t to total mutants of s. We then calculated, for each test case t_k in T, an award value for t_k, by summing all ms(s; t_k) values. Total fault-exposing-potential prioritization orders the test cases in a test suite in order of these award values.

This is an approximation method; FEP prioritization is more costly than code-coverage-based techniques due to the expenditure of mutation analysis. If FEP prioritization shows promise, however, this would motivate a search for cost-effective approximations of fault-exposing potential.

Additional FEP prioritization

Similar to the extensions made to total statement coverage prioritization to obtain additional statement coverage prioritization, total FEP prioritization is also extended to generate additional fault-exposing-potential (FEP) prioritization. In case of additional FEP prioritization, a test case t is selected initially, then it award values for all other test cases are lowered that exercise statements in the correctness of those statements; then selection of a next test case is made, repeating this process until all test cases are sorted.

4.4.3 Function Level Techniques

Total function coverage prioritization

Equivalent to total statement coverage prioritization but dealing with the level of functions, this technique prioritizes test cases on the basis of the total number of functions which are executed.

Additional function coverage prioritization

Equivalent to additional statement coverage prioritization but dealing with the level of functions, this technique prioritizes test cases on the basis of the total number of additional functions which are covered.

Total FEP (function level) prioritization

This technique is similar to total FEP prioritization at the statement level. To transform that technique to the function level, a function level approximation of fault-exposing potential is required. Then mutation analysis is used for

computing each test case t and each function f, the ratio of mutants in f exposed by t to mutants of f executed by t. Adding up these values, award values for test cases are obtained. Then same prioritization algorithm is applied as for total FEP (statement level) prioritization, replacing functions with statements.

Additional FEP (function level) prioritization
In this technique, the total FEP (function level) technique is extended in the same manner the total FEP (statement level) technique is extended.

Total fault index (FI) prioritization
Faults are not equally expected to exist in each function; rather, certain functions are more liable to contain faults than others. This fault proneness can be associated with measurable software attributes. Test cases based on their history of executing fault prone functions are prioritized taking the advantage of their association. To represent fault proneness, a fault index based on principal component analysis is used **[23]**.Generation of fault indexes requires measurement of each function in the new version, generation of fault indexes for the new version, and comparison of the new indexes against the indexes calculated for the baseline version. Each function is then assigned an absolute fault index representing the fault proneness for that function, based on the complexity of the changes that were introduced into that function. By these fault indexes, total fault index coverage prioritization is performed in a manner analogous to total function coverage. For each test case, the sum of the fault indexes are computed for every function that test case executes. Then, the test cases are ordered in decreasing order of these sums.

Additional fault-index (FI) prioritization
Additional fault index coverage prioritization is accomplished in a manner analogous to additional function coverage. The set of functions are covered by earlier executed test cases is maintained. If this set contains all functions. The mechanisms of the method are given in **[23]**. To find the next best test case for each test case, the sum of the fault indexes for each function that test case executes, except for functions in the set of covered functions are computed. The test case for which this sum is the greatest will win. This process is repeated until all test cases have been prioritized.

Total FI with FEP coverage prioritization
A superior rate of fault detection is obtained by using both an approximation of fault exposing potential and an estimate of fault proneness. Therefore, in this technique, first total fault index prioritization to all test cases is applied; then, for all test cases that have equal fault index award values, apply total FEP prioritization as a secondary order.
Additional FI with FEP coverage prioritization
The previous technique to an additional" variant is extended. In this technique, additional fault index prioritization is used to achieve an initial test case ordering; then apply FEP prioritization to rank all test cases possessing equal fault-index-based award values.

4.5 Distribution Based Techniques

Leon et al. introduced distribution-based filtering and prioritization **[24]**. Distribution-based techniques reduce and prioritize test cases on the basis of the distribution of profiles of test cases in the multi-dimensional profile space. Test case profiles are produced by the dissimilarity metric, a function that produces a real number which represents the degree of dissimilarity between two input profiles. Using this metric, test cases can be clustered or divided into classes having the similar profiles according to their properties. The clustering can reveal some interesting information. For example:

- Clusters of similar profiles may indicate a group of redundant test cases
- Isolated clusters contain test cases suggest unusual conditions that are perhaps expected to cause failures
- Low density regions of the profile space may indicate uncommon usage behaviors

The first point is related to reduction of effort; if test cases in a cluster are indeed very similar, it is sufficient to execute only one of them. The second and third points are related to fault-proneness. Certain unusual conditions and uncommon behaviors may tend to be harder to reproduce than more common conditions and behaviors. Therefore, the corresponding parts of the program are likely to be tested less than other, more frequently used parts of the program. Assigning a high priority to test cases that execute these unusual behaviors may increase the chance of early fault detection. A good example might be exception handling code.

4.6 Hybrid Approaches

Hybrid Approaches are the techniques formed by combining features of other techniques and by achieving different objectives through various techniques used in a combination. In this, two or more approaches can be combined to attain the two or more goals. For example, combining MCEFD and Customer requirement based techniques; both functional and structural testing can be done for any system. Customer requirements are used for system design, thus, important phases of SDLC that is, design and coding are tested completely in this example. Leon et al. developed new prioritization techniques that combine coverage-based prioritization with distribution-based prioritization. This hybrid approach is based on the observation that basic coverage maximization performs reasonably well compared to repeated coverage maximization **[24]**. The authors observed that the fault detection rate of repeated coverage maximization is not as high as that of basic coverage maximization. This motivated them to consider a hybrid approach that first prioritizes test cases based on coverage, then switch to distribution-based prioritization once the basic coverage maximization is achieved. They considered two different distribution-based techniques. The one-per cluster approach samples one test case from each cluster or class, and prioritizes them according to the order of cluster creation during the clustering. The failure-pursuit approach behaves similarly, but it adds the closest neighbors of any test case that finds a fault. The results showed that the distribution-based prioritization techniques could outperform repeated coverage maximization **[24]**.

4.7 Other Approaches

Rothermel et al. analyzed the use of mutation score for test case prioritization along with other structural coverage criteria **[17]**. Hou et al. considered interface contract mutation for the regression testing of component-based software and evaluated it with the additional prioritization technique **[25]**. Sampath et al. presented the prioritization of test cases for web applications **[12]**. The test cases are, in this case, recorded user sessions from the previous version of the SUT. Session-based test cases are thought to be ideal for testing web applications because they tend to reflect the actual usage

patterns of real users, thereby making for realistic test cases. They compared different criteria for prioritization such as the number of HTTP requests per test case, coverage of parameter values, and frequency of visits for the pages recorded in sessions and the number of parameter values. The empirical evaluations showed that prioritized test suites performed better than randomly ordered test suites, but also that there is not a single prioritization criterion that is always best. However, the 2-way parameter-value criterion, the prioritization criterion that orders tests to cover all pair-wise combinations of parameter-values between pages as soon as possible, showed the highest APFD value for 2 out of 3 web applications that were studied.

Fraser et al introduced a model-based prioritization approach **[14]**. Their prioritization technique is based on the concept of property relevance **[13]**. A test case is relevant to a model property if it is theoretically possible for the test case to violate the property. The relevance relation is obtained by the use of a model-checker, which is used as the input to the greedy algorithm. While they showed that property-based prioritization can outperform coverage-based prioritization, they noted that the performance of property-based prioritization is heavily dependent on the quality of the model specification. A few techniques and analyses used for test suite minimization or regression test selection problem have been also applied to test case prioritization. Rummel et al. introduced a prioritization technique based on data-flow analysis by treating each du pair as a testing requirement to be covered **[26]**.

5. CONCLUSION

Test case prioritization is a way to prioritize and schedule test cases. The technique is developed so as to run test cases of higher priority in order to minimize time, cost and effort during software testing phase. The literature review shows that many researchers have proposed many methods to prioritize and reduce the effort, time and cost in the software testing phase; such as test case prioritization methods, regression selection techniques and test case reduction approaches. This paper concentrates on test case prioritization techniques only. This study introduces a new classification scheme for test case prioritization called as “3CMDHO”.

6. REFERENCES

[1] R. Pressman, Software Engineering: A Practitioner's Approach.: Mc-GrawHill, 2005.

[2] [tutorialpoint.Online]. www.tutorialspoint.com/software_testing

[3] S. Elbaum, A. Malishevsky, G.Rothermel, "Test case prioritization: A family of empirical studies," IEEE Transactions on Software, Febrary 2002.

[4] Aditya P.Mathur, Foundation of software testing, 1st ed.: Pearson.

[5] Maruan Khoury, Cost-Effective Regression Testing., 2006.

[6] W.Wong, J. Horgan, S. London, H. Agrawal, "A study of effective regression testing in practice," in Eighth Intl. Symp. on Softw. Rel. Engr, 1997, pp. 230-238.

[7] Gregg Rothermel,Roland H. Untch,Mary Jean Harrold, "Prioritizing Test Cases For Regression Testing," IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, vol. 27, no. 10, October 2001.

[8] Sebastian Elbaum,Alexey G. Malishevsky,Gregg Rothermel, "Prioritizing Test Cases for Regression Testing," International Symposium of Software Testing and Analysis, pp. 102-112, August 2000.

[9] J. Jones, M. Harrold., "Test-suite reduction and prioritization for modified condition/decision coverage," in International Conference on Software Maintenance, 2001.

[10] Srivastava, J. Thiagarajan, "Effectively prioritizing tests in development environment," in International Symposium on Software Testing and Analysis, 2002, pp. 97-106.

[11] Avritzer and E. Weyuker., "The automatic generation of load test suites and the assessment of the resulting software," IEEE Transactions on Software Engineering, vol. 21, no. 9, pp. 705-716, December 1995.

[12] Sampath S, Bryce RC, Viswanath G, Kandimalla V, Koru AG, "Prioritizing user-session-based test cases for web applications testing," in 1st International Conference on Software Testing Verification and Validation, 2008, pp. 141-150.

[13] Fraser G,Wotawa F, "Property relevant software testing with model-checkers," SIGSOFT Software Engineering Notes , vol. 31, no. 6, pp. 1-10, 2006.

[14] Fraser G,Wotawa F, "Test-case prioritization with model-checkers," in 25th conference on IASTED, USA, 2007, pp. 267-272.

[15] Kim, J. M.; Porter, A, "A history-based test prioritization technique for regression testing in resource constrained environments," in 24th International Conference on Software Engineering, 2002.

[16] Wong WE, Horgan JR, London S, Mathur AP, "Effect of test set minimization on fault detection effectiveness," Software Practice and Experience, pp. 347-369, April 1998.

[17] Gregg Rothermel, R. H. Untch, C. Chu, and M. J.Harrold, "Test case prioritization: An empirical study," in IEEE International Conference on Software Maintenance, UK, 1999, pp. 179-188.

[18] Alexey G. Malishevsky, Gregg Rothermel,Sebastian Elbaum, "Modeling the Cost-Benefits Tradeoffs for Regression Testing Techniques," in International Conference on Software Maintenance, 2002.

[19] Jung-Min Kim, Adam Porter, Gregg Rothermel, "An Empirical Study of Regression Test Application Frequency," International Conference on Software Engineering, 2000.

[20] Hema Srikanth, Laurie Williams, Jason Osborne, "System Test Case Prioritization of New and Regression Test Cases," in 4th International Symposium on Empirical Software Engineering, 2005, pp. 62-71.

[21] M. Thompson, D. Richardson, L. Clarke, "An information flow model of fault detection," Int'l. Symp. on Softw. Testing and Analysis, pp. 182-192, 1993.

[22] R. G. Hamlet, "Testing programs with the aid of a compiler," pp. 279-290, July 1977.

[23] S. G. Elbaum and J. C. Munson, "Code churn: A measure for estimating the impact of code change," in Int'l. Conf. Softw. Maint, 1998, pp. 24-31.

[24] Leon D, Podgurski A, "A comparison of coverage-based and distribution-based techniques for filtering and prioritizing test cases," in IEEE International Symposium on Software Reliability Engineering, 2003, pp. 442-456.

[25] Hou SS, Zhang L, Xie T, Mei H, Sun JS, "Applying interface-contract mutation in regression testing of component-based software," in 23rd IEEE International Con

[26] Reference on Software Maintenance, 2007, pp. 174-183.Rummel M, Kapfhammer GM, GM, Thall, "Towards the prioritization of regression test suites with data flow information," in 20th Symposium on Applied Computing, 2005.