International Journal of Recent Technology and Engineering (IJRTE)
ISSN: 2277-3878, Volume-8 Issue-4, November 2019

Importance of an Effective Test Suite Minimization Technique in Software Testing

Fayaz Ahmad Khan

Abstract: ***Software testing is an important activity for developing a quality end product. But it is expensive as it requires both time and cost to perform it during development phase and after its delivery. Therefore, it is required to develop some techniques in this direction in order to make software testing a desirable as well as an affordable activity. Hence, in this study we have proposed an effective method that minimizes the size and redundancy from an automated random generated test suite using data clustering technique. The proposed technique compared to other existing techniques achieves a substantial amount of reduction in the size of the test suite without affecting the fault detection capability of the test suite.***

Keywords: ***Software Engineering. Software Testing, Test suite, Test Suite Minimization and Data Clustering***

I. INTRODUCTION

Software testing is a pervasive activity in software development. Testing is widely used to reveal bugs in real software development and is also an expensive task. To reveal bugs, software testing and retesting occurs continuously during and after software development. Typically, a test suite should be prepared before initial testing is started. As software evolves and grows new test cases are added to the test suite. Over time, some test cases in the test suite become redundant as the requirements covered by them are also covered by the other test cases [1, 2]. To reduce the cost, time and effort used to carry out testing, it is very important to develop some techniques that will help in keeping the size of the test suite to a manageable number and will eliminate the redundant test cases from it. The reduction or minimization can be achieved at the time of generating the test cases or after acquiring an initial test suite in case of regression testing. In literature [1] [2] many efforts have been put forward to address the issue of redundancy in the test suites by taking only one testing criterion in consideration. A test case or a test suite is created to exercise certain requirements of the software under test. A requirement that needs to be exercised or covered by a test case may be either white-box (concerned to the code itself) or black-box (dealing with the specifications).

Manuscript published on November 30, 2019.
* Correspondence Author
Dr. Fayaz Ahmad Khan*, Department of computer science & applications Barkatullah University Bhopal.

Various types of test criterions exists that describe the degree to which a source code or unit under test has been tested, for example: statement coverage criterion, decision coverage, branch coverage [2] [3], path coverage and other requirement coverage criterions [1] [4]. A test adequacy criterion also provides a measurement of test suite quality and guides in test data generation. A tester can set any of the aforesaid criteria as testing criteria to ensure the complete testing of the application.

Software testing plays a very important role in assuring the quality and reliability of the software under test. As the size and complexity of the under developed product grows, the time and effort required for effective testing also increases. Literature on software testing indicates that more than 50% of the cost of software development is devoted to testing [5]. One of the important issues in software testing is the test case generation. Test case design or generation is time consuming and labor intensive task. There are usually two approaches followed while designing test cases; one by manually and second by using automation techniques [5]. The manual way of designing is very time consuming and error prone. To save time and reduce the effort, automated techniques are used for test case generation. But due to automation in the test case generation, large and redundant test cases are generated which take longer time for execution. Test cases become redundant with respect to a specified test requirement as the requirements exercised by those redundant test cases are also exercised by other test cases present in the test suite. Thus, it is very important to develop some new efficient techniques that will minimize the number of test cases in a test suite in order to decrease the time and cost devoted in software testing.

II. TEST SUITE MINIMIZATION AND RELATED WORK

Test objectives or requirements are usually defined before the software is tested and are very different from each other and may also vary in granularity. A test case requirement can be defined with a small granularity like the coverage of the every statement and on the other hand, it can be defined with large granularity, such as the coverage of every user requirement. As one test case is not sufficient to satisfy all the user requirements, it usually requires large number of test cases to satisfy as many as possible test requirements [6]. Thus in practice, the test suite undergoes the process of expansion due to the addition of new test cases as and when the software is modified. The other reason behind the size of test suite is the input domain of program variables which is very large and exhaustive testing with whole domain is not adequate for practical execution.

Thus, it has long been identified as a research problem to find a minimized subset of test cases from an initially automatically generated test suite that will satisfy same set of test requirements as satisfied by it before minimization for an effective and efficient testing [6]. This problem is usually referred to as test suite minimization problem. In [6], test suite minimization is defined as:

Definition: Given T (a test suite) and R (a set of test requirements r1, r2, r3. . . r_i) that must be satisfied to get the desired coverage of the program.

Problem: Find a subset T' **of** T that satisfies all r_i^s**.**

In literature, various approaches have been proposed in the area of test suite minimization. Most of the research work carried out on the test suite minimization relied on heuristics for determining near-optimal solutions.

Jones and Harrold [7] proposed two minimization heuristics algorithms and have used modified condition/ decision coverage criterion as test adequacy criteria. One algorithm is used to create minimized test suite by identifying incrementally the essential and redundant test cases and the second algorithm make use of prioritization techniques to stop computing before all the test cases have been prioritized. Another work on test suite minimization using the notions of mega blocks and global dominator graphs is proposed by Agrawal in [8]. A noteworthy contribution in the area of test suite minimization is given by Harrold, Gupta, and Soffa [6] that attempts to minimize test suite on the basis of a given set of program requirements. The approach uses a greedy heuristic to choose the next test case that covers the requirement which is next-hardest to satisfy by the test suite and continuing in this way until all requirements covered by the minimized test suite. The results of the study conducted by the authors showed a significant reduction in the size of test suite. Rothermel et al. [9] conducted a series of experiments into the test suite minimization based on the algorithm proposed by Harrold, Gupta, and Soffa [6]. The authors in [9] argued that contrary to the results suggested in previous work [10, 11, 12], test suite minimization approaches can severely compromise the fault detection effectiveness of minimized test suites. Jeffrey and Guppta [13], proposed a modified form HGS algorithm [6] known as Reduction with Selective Redundancy in order to retain certain redundant test cases to ensure that reduction did not compromise the fault detection ability of the minimized test suite. The proposed algorithm deals with the limitations of traditional single-criterion minimization techniques by taking in account several sets of testing demands (e.g., coverage of different entities) and introducing selective redundancy in the minimized test suites. Simran et al. in [14] proposed a delayed greedy algorithm using concept analysis. Reduction in test suite is also achieved by reducing the requirement set using graph retraction techniques [15]. In [1], call tree construction approach is proposed to address the test suite minimization for white box testing. But the construction of call tree is very cumbersome process. Also in [16], an approach based for embedded nondeterministic systems based on testing in context is also proposed. Mutation analysis is also an important technique used for validating the test suite by determining its effectiveness in terms of the code coverage characteristics [17]. But, generation of mutants in mutation analysis is a very difficult task that limits its scope in real applications. The other useful techniques used for test suite minimization are, techniques based on genetic algorithm [18] and Integer linear programming [4].

III. PROPOSED APPROACH AND ITS MOTIVATION

Software testing is mainly carried out at different levels and such levels namely include unit testing, integration testing, system testing and acceptance testing. The first three techniques in first three levels are carried out by tester during the different stages in software development life cycle. The last technique or last level of testing (acceptance) is done by the users or customers. Unit testing is one of the widely used techniques for the assurance of high quality of the software product. The authors in [19] defined a software unit as:

> "A software unit is the smallest testable piece of software and it may consist of hundreds or even just few lines of source code, and generally represents the result of the work of one or few developers."

The main objective associated with unit testing is to achieve high or acceptable code coverage such as statement coverage, branch coverage, path coverage and data flow coverage when executed with a set of test cases. The goal of testing is to uncover as many as faults in a unit under test with a potent set of test cases. To generate such a potent set of test cases automatically is a very difficult task and also an intellectually demanding research area in present software industries [20, 21, 22, 23, 24, 25]. The fact behind is that the domain of input variables is extremely very large and the choice of exhaustive testing is impossible as well as impractical due time and resource constraints. For example, a simple program with three input variables (A, B and C) can have $(2^{16} \times 2^{16} \times 2^{16}) = 2^{48} \approx 256 \times 10^{12}$ possible assignments to the input triple (A, B, C). To exhaustively test (test using all combinations of input data) it will take more than eight years if we do testing 24 hours a day, 7 days a week. The only way or option is to use a part of the input domain for testing not the whole domain. The further questions arises here is that which values and how many needs to be selected to compose a test suite and maximize the chance of detecting the faults. In literature many such automatic test data generation techniques like random [23], Symbolic execution [24] and search based [25] techniques have been proposed and have revealed promising results. But due to automation in the test case generation process still large and redundant test cases are generated which take longer time for execution. Therefore, depending on the magnitude of the model, these algorithms or techniques produce a significant number of test cases that are infeasible to be considered for practical execution. Therefore, other than structural coverage criteria, test-case generation process may need to be combined with selection strategies that may help to focus test generation at particular functionalities of interest; minimize redundancy in test suites; and limit the size of test suites. Hence, it is very important to develop techniques that will minimize the number of test cases in a test suite order to reduce the time and cost devoted in testing. Most of the existing minimization techniques [16, 17, 18] have achieved high test suite size reduction by removing certain test cases with respect to a single coverage criterion, but at the expense of severe fault detection loss.

Because removing certain redundant test cases permanently from a test suite according to a single adequacy criterion will throw away certain very useful test cases that are not redundant with respect to other adequacy criterions. So according to our understanding, we need to develop some useful techniques that will minimize the size of an existing test suite with respect to multiple adequacy criterions or using some other criteria so that the fault detection ability of the minimized test suite is preserved for an effective and efficient testing.

In this study, we therefore propose an efficient technique for test suite minimization using data clustering technique. Cluster analysis an important domain of data mining, groups the data into meaningful groups/ clusters based on the information found in the data [26,27,28]. The aim is that the objects in a cluster are related to one other and separate or different from the objects in other clusters [26,29,30]. The purpose of using clustering technique serves two important objectives with respect to test suite minimization perspective; in one way, our techniques reduced the input domain (test suite) into a fixed number of partitions or domains, and in second way, they perform grouping of redundant test cases into their appropriate partitions or domains based on the similarities found in the test cases [28]. One of the other important characteristics of our proposed approach is that the fault detection effectiveness of the test suite is preserved, because it does not throw away or eliminate completely redundant test cases from the test suite, but groups them in their appropriate clusters. All the existing techniques [6, 7, 8, 9, 10] perform minimization based on certain code coverage criterions or taking the code coverage matrix into consideration, but our approach is entirely independent of any code coverage criteria. It minimizes the test cases based on the similarity that exists between the test cases and then partitions the entire test suite into different partitions. The other useful characteristics of the proposed approach is that it can not only be applied for regression testing, but also after test data generation process in order to limit the size of the generated test cases. The steps in our proposed approach are graphically represented in figure 1 below.

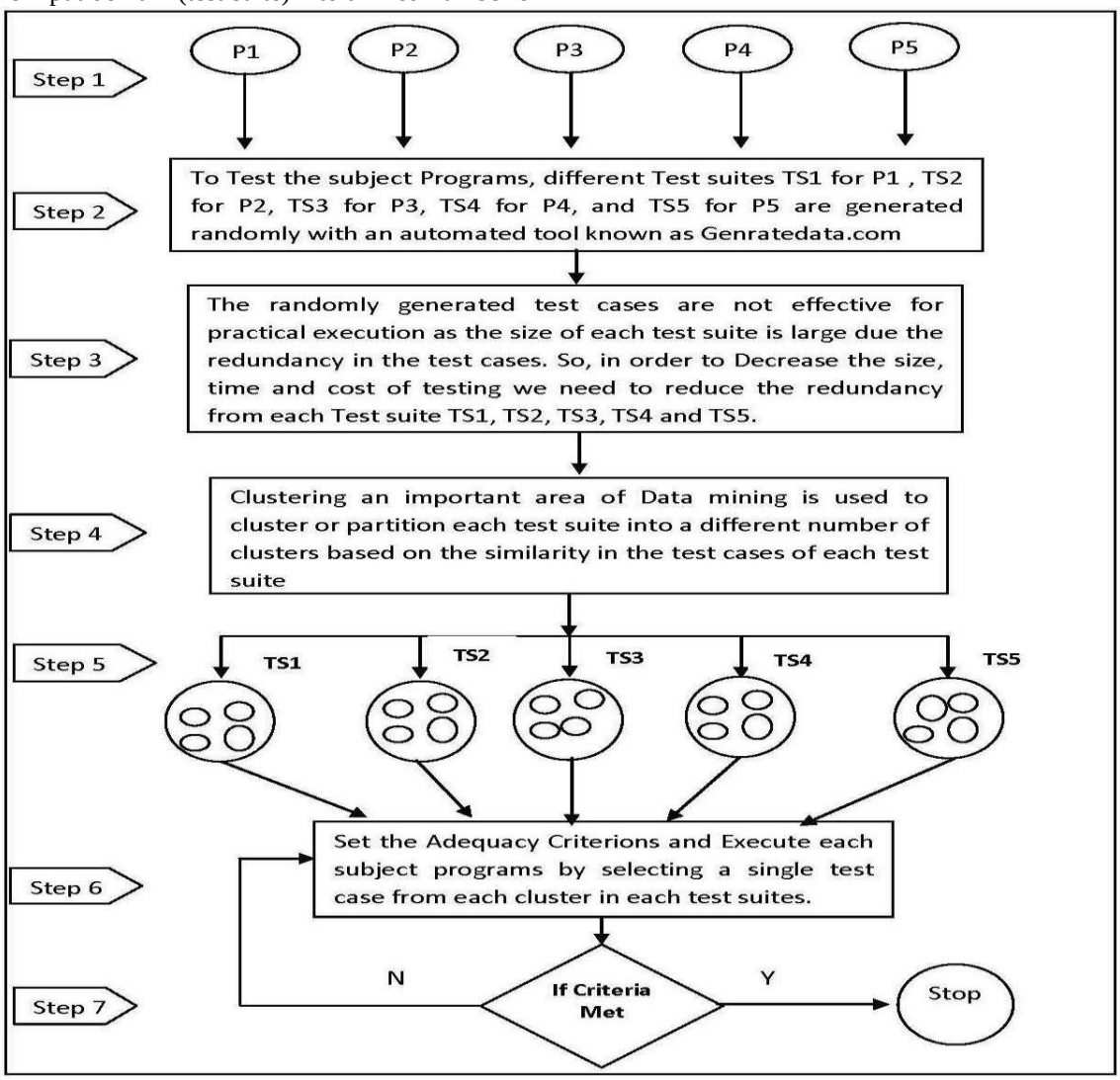

Fig. 1. Steps of the Proposed Approach

IV. EXPERIMENTAL ENVIRONMENT

The goal of our proposed test suite minimization technique is to reduce both the size and redundancy from a randomly generated test suite for an effective and efficient testing. To carry out the experimental study we have used Eclipse Java Neon framework as an experimental environment with EclEmma and JUnit tool as a plug-ins for the code coverage measurement and automatic test case execution.

The initial test suite for testing the subject programs are generated with an open source tool known as generatedata.com. The tool randomly generates a pool of test cases within a specified range. The experiment is conducted on an Intel® Core i5 PC with 4GB physical RAM and running Windows 7 Home Basic operating system.

A. Subject Programs

For a successful implementation of our proposed approach, we have used the well-known collection of programs shown in table 1, as our experimental subjects.

Table- 1. Experimental Subject Programs

S No.	Subject Programs	Range of Variables	Total Test cases
P1	Given Number is Prime or Not	-10 to 110	100 rows
P2	Largest from given Numbers	-10 to 110	100 rows
P3	Triangle Classification	-10 to 110	100 rows
P4	Digits in a given Number	-10 to 110	100 rows
P5	Leap Year	1980 to 2015	35 cases

B. Test Suite Generation and Minimization

To test the subject programs, we generated a different pool of test cases within a specified range as shown in table 1, with an automated tool known as generatedata.com [31]. The tool [31] randomly generates test cases within the specified range. The range is important to specify, because exhaustive testing with whole input domain is infeasible and impractical in real settings. Random approach is simple and cost effective way of generating test data, but it has some limitations that are, it generates large and redundant test cases which satisfy same requirements multiple times. The other important drawback that limits the scope of the random approach is that, it is not capable of generating a few important combinations of test cases within the specified range. The inefficiency of not generating few combinations of test cases is addressed in chapter 3, here in this chapter; we deal with the issues of redundancy and the size of the generated test suites. It is important to note that, we are minimizing the test case before regression testing so that the cost of regression testing is minimized by removing redundant test cases from the test suites. All the existing test suite minimization techniques minimize the size of the test suites based on certain code coverage criterions, but through the proposed study we minimize the test suite size based on the similarity that exists between the test cases not on the basis of any test adequacy criterion. To minimize the size and redundancy from each automatically generated test suite for testing each subject programs described in table 1, we implemented hierarchical clustering algorithm in Weka software, to cluster them into a different specified number of clusters. Weka is open source software that contains a collection of machine learning algorithms for data mining tasks. The algorithms can either be used directly on test data or called from own java code. The procedure used to partition each test suite is graphically shown in figure 2. But, it also very important to note and verify that, are these generated test cases capable of achieving the desired code coverage characteristics of the subject program under study. To evaluate the effectiveness of the minimized test suites, test metrics are used and are discussed in the following section.

C. Test Metrics for Performance Evaluation

Test metrics are used as performance evaluators during software testing. The measurements of testing process attributes enable a software tester to have a better understanding into the software testing process. The following few metrics were set or used to evaluate the performance of the proposed technique.

1. Test suite size reduction or percentage of reduction. It is measured as:

$$T\_reduction = \frac{(\text{Initial size } (T) - \text{Size after reduction } (T_{rs}))}{\text{Initial Size } (T)} \times 100$$

$$= ((T - T_{rs}) \times 100) / T$$

2. Percentage of Requirement Coverage is calculated as:

$$R\_Coverage = \frac{(\text{Requirements covered by the minimized test suite})}{\text{Total Requirements}} \times 100$$

D. Experimental Results and Discussion

Agglomerative hierarchical clustering technique with average linkage proximity measure is implemented on each test suite of sample programs listed in table 1. Each test suite (TS1, TS2, TS3, TS4 and TS5) is converted into CSV format in order to apply clustering algorithms in Weka. For applying clustering algorithms, we need also to determine the number of clusters (the value of k) before implementation. In this study, we also propose the value of k (the number of partitions) for each test suite and is calculated based on the number of requirements that must at least be satisfied by each test suite. The table 2 depicts the minimum number of requirements that needs to be satisfied by each test suite and the corresponding value of k. The size and redundancy aspects of test suites are two critical issues in software testing process and many technique or algorithms exist that in some way proved beneficial in certain circumstances.

Retrieval Number: D6885118419 /2019©BEIESP
DOI:10.35940/ijrte.D6885.118419
Journal Website: www.ijrte.org

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

But there is no concrete solution to these problems, because finding the optimal representative set from the test suite has been identified as an NP-Complete problem [32].

Table- 2. Minimum Requirements and corresponding value of K

Subject Program	Requirements that must be satisfied	The proposed value of K
P1	1. No. is Prime 2. No. is not Prime 3. Invalid Input 4. Outside the valid Range	4
P2	1. A is grater 2. B is greater 3. C is greater 4. Equal Numbers 5. Invalid Input	5
P3	1. Scalene 2. Isosceles 3. Equilateral 4. Not a triangle 5. Invalid Input	5
P4	1. Number of digits 2. Invalid Input	2
P5	1. Leap Year 2. Not a Leap Year 3. Invalid Date	3

After settings the necessary parameters for carrying out clustering, the results of clustering implemented on the test suites (TS1, TS2, TS3, TS4, and TS5) are depicted in figure 3, figure 4, figure 5, figure 6, and figure 7.

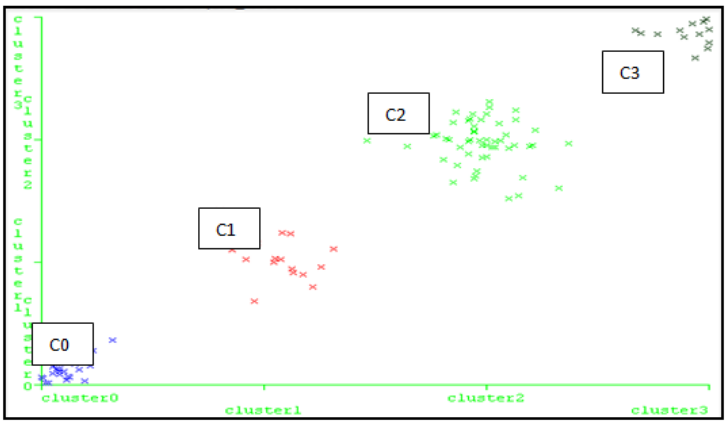

Fig. 3. Segments of TS1 after Clustering

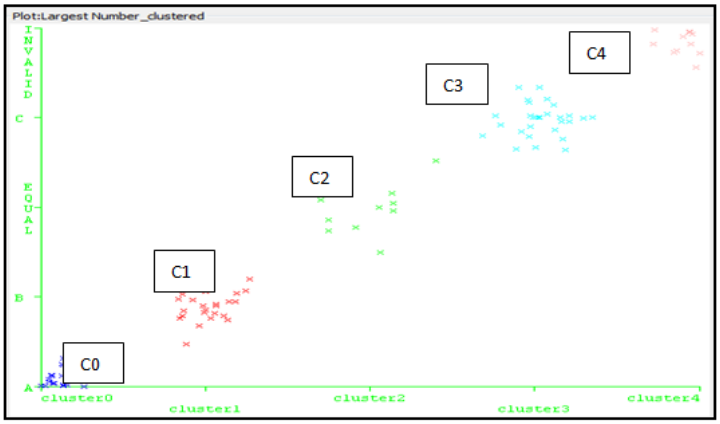

Fig. 4. Segments of TS2 after Clustering

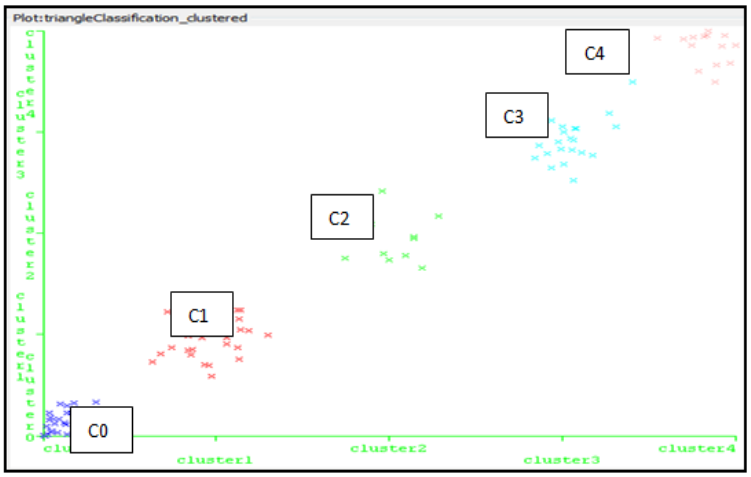

Fig. 5. Segments of TS3 after Clustering

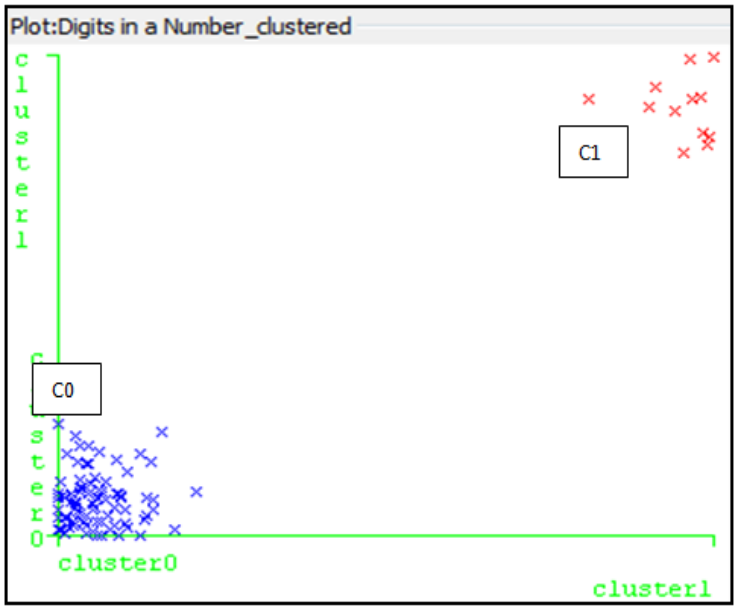

Fig. 6. Segments of TS4 after Clustering

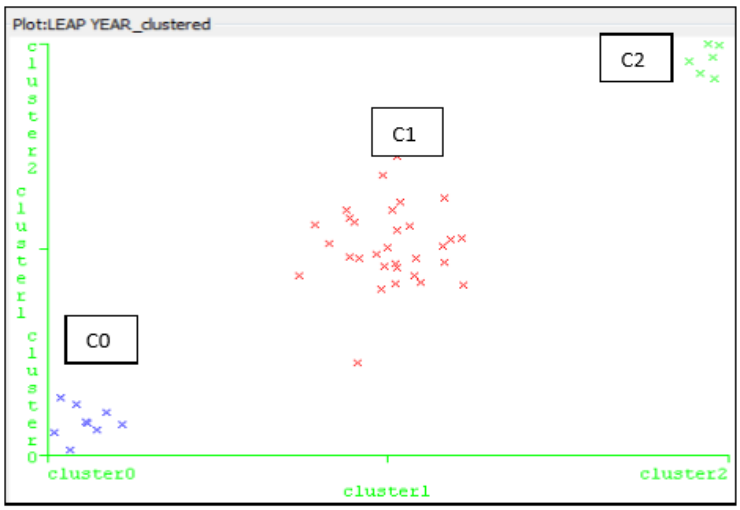

Fig. 7. Segments of TS5 after Clustering

After partitioning each test suite (TS1, TS2, TS3, TS4 and TS5) into a specified number of clusters or domains using clustering technique, we then from a new minimized test suite (MTS1 for P1, MTS2 for P2, MTS3 for P3, MTS4 for P4, MTS5 for P5) for each subject program by selecting a single test case from each cluster in each partitioned test suite for execution. This step is graphically represented in figure 8.

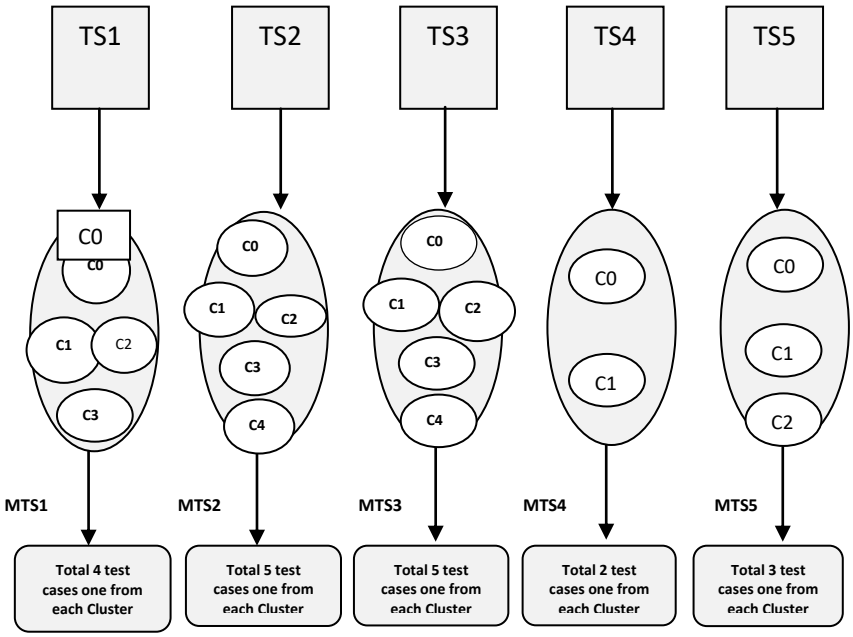

Fig. 8. Selection and Formation of New Minimized Test Suites

The reduction in the number of test cases achieved with the proposed approach is significant and is shown in figure 9. The minimized test suites (MTS1, MTS2, MTS3, MTS4, and MTS5) for each subject programs after implementation of the proposed approach at any instant contain only 4 test cases for testing P1, 5 test cases for testing P2, 5 test cases for testing P3, 2 test cases for testing P3 and 3 test cases for testing P5. The other important characteristic of our proposed approach is that, we have not eliminated the test cases from the test suite but we at a particular instant consider only few of them for the execution. The inclusion of more test cases will depend on the code coverage characteristics achieved by the initial seed of test cases present in each minimized test suite. If the code coverage or requirement coverage is poor, we need to select new seed or population of test cases from each cluster in each partitioned test suites for execution for better coverage. The comparison between the initial test suites size and the final size after minimization is shown in figure 9.

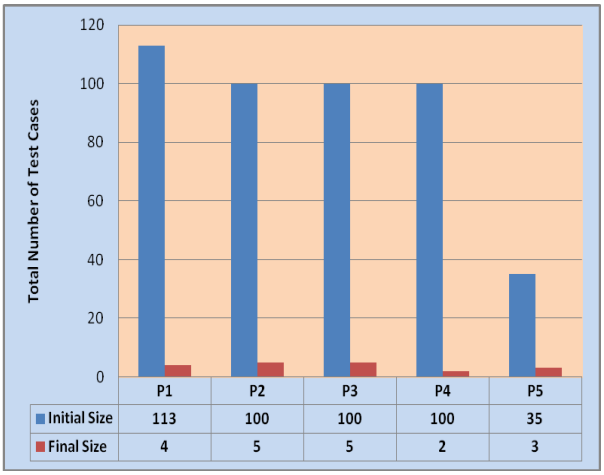

Fig.9. Test Suite Size Comparison

The percentage of reduction in the number of test cases achieved with the proposed approach is also very significant and is tabulated in table 3 below:

Table- 3. Percentage of Reduction Achieved

Subject Programs	Initial Test suite Size (T)	Size after Minimization (T_{rs})	% of size Reduction
P1	113	4	**97.3%**
P2	100	5	**95%**
P3	100	5	**95%**
P4	100	2	**98%**
P5	35	3	**91.4%**

The other important metric used for test suite evaluation is the percentage of code coverage or requirement coverage achieved by the minimized test suite. For the present study we have used statement coverage and branch coverage as test criterion for test suite evaluation.

After execution of the test cases from the minimized test suite the percentage of requirement coverage achieved by each minimized test suite for the study is shown below: The percentage of requirement coverage achieved by the minimized test suite (MTS1) in case of subject program P1 is shown in figure 10.

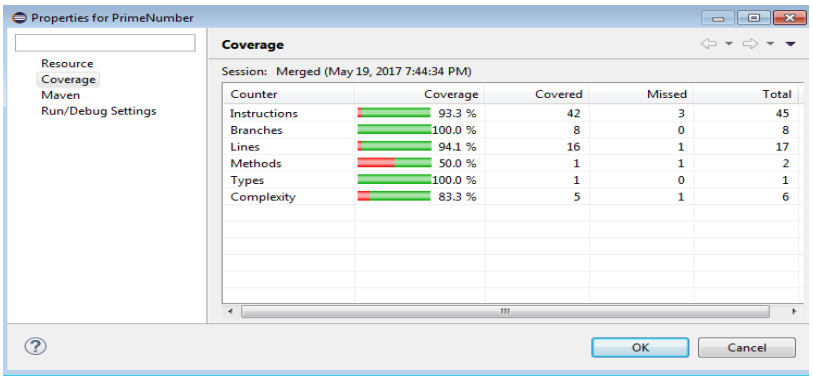

Fig. 10. Requirement Coverage achieved by MTS1 after execution on P1.

The requirement coverage gained by minimized test suite (MTS2) after execution on the subject program P2 is shown in figure 11.

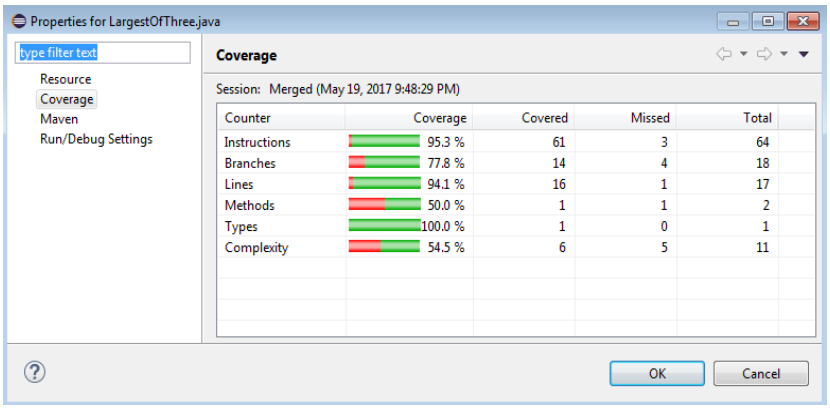

Fig. 11. Requirement Coverage achieved by MTS2 after execution on P2.

The percentage of requirement coverage achieved by the minimized test suite (MTS3) in case of subject program P3 is shown in figure 12.

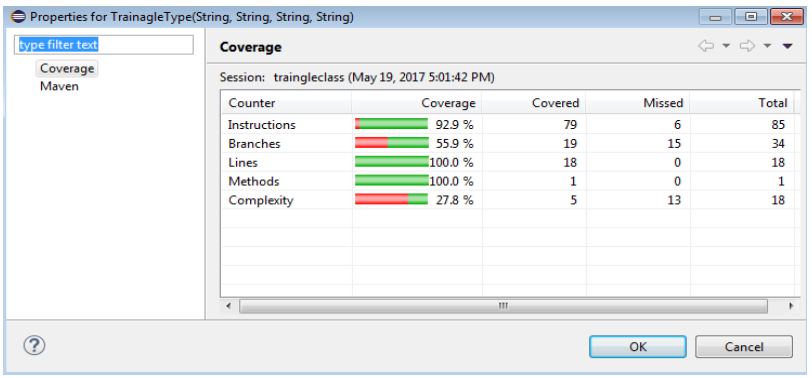

Fig. 12. Requirement Coverage achieved by MTS3 after execution on P3

The requirement coverage gained by minimized test suite (MTS4) after execution on the subject program P4 is shown in figure 13.

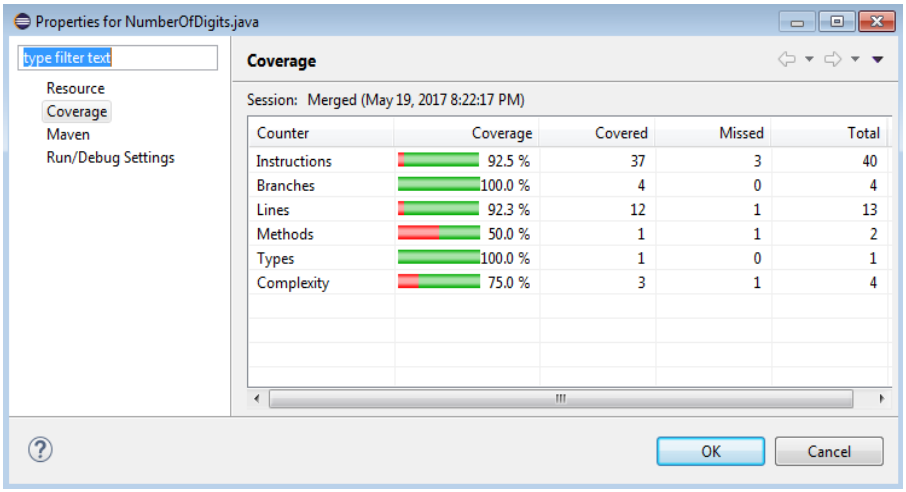

Fig. 13. Requirement Coverage achieved by MTS4 after execution on P4

The code coverage characteristics achieved by the minimized test suite (MTS5) after executing on the subject program P5 is shown in figure 14.

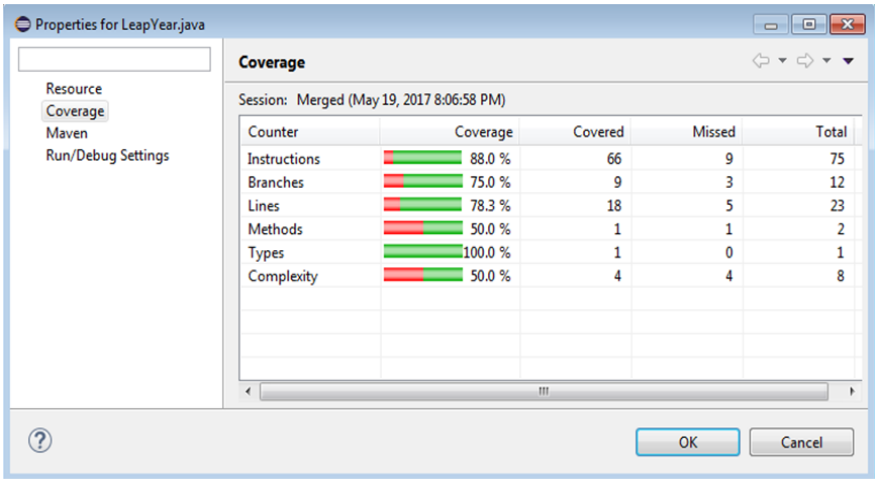

Fig. 14. Requirement Coverage achieved by MTS5 after execution on P5

E. Comparison with other Approaches:

The focus of our study and the experimentation is to compare the minimization or reduction results achieved by our technique with the existing techniques. From the existing techniques we choose HGS [6], BOG [33] and MFTS [34] techniques. The Harrold, Gupta and Soffa (HGS) technique [6] follows a heuristic to greedily select the test cases from the test requirement matrix in order to form a representative set of test cases of smallest size covering the same requirements as covered by the original test suite. The test case requirement matrix shows the relationship between a test cases and testing requirements. The Bi-Objective Greedy (BOG) algorithm [33] also takes into consideration the multiplied coverage matrix (test case requirement matrix multiplied by its transposed matrix) and then greedily selects optimum test cases until a same coverage of the requirements is resulted. The maximum Frequent Test Sets (MFTS) algorithm [34] uses data mining technique to address the issue of test suite minimization. The MFTS algorithm [34] selects maximum frequent test sets based on the support value to generate an optimal test suite. The proposed approach based on data clustering technique is not based on any code coverage criterion and thus does not require a requirement matrix compared to most of the existing techniques [6, 33, 34] for the determination of representative test cases. It utilizes the proximity matrix as a measure of similarity between the test cases for grouping the test cases into the specified number of clusters. The percentage (%) of test suite size reduction achieved by the proposed approach in comparison with the existing approaches is depicted in figure 15. The results in figure 15 indicate that a significant percentage of reduction (**Above 90%** along Y-axis) in the number of test cases for the subject programs (along x-axis) is achieved by the proposed approach against the existing approaches [6, 33, 34].

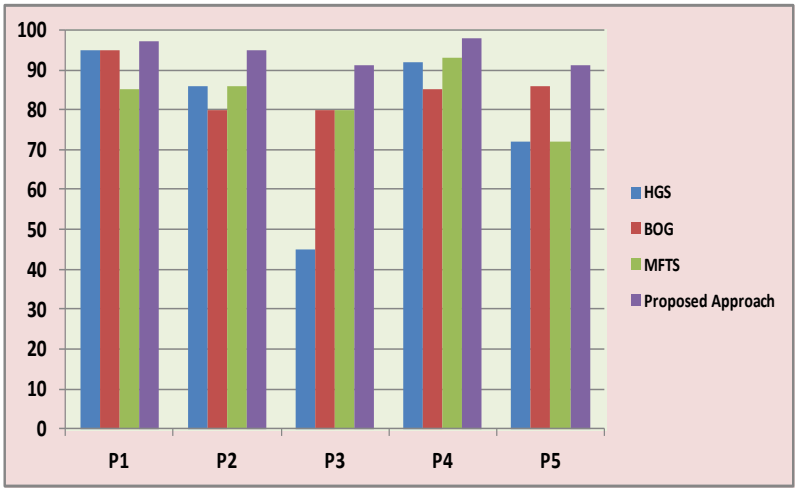

Fig. 15. Percenatge of Reduction in Test Suite Size

To access the quality of the minimized test suites we have also determined the percenatge of requirement coverage obatained by the proposed approach with respect to existing approaches. The comaprision of percentage of code coverage between our approach and the existing approaches is shown in figure 16. The average percenatge of code coverage achieved by the our approach as shown in figure 16 is arround 86% which is less then MFTS [34] but higer then HGS [6] and BOG[33]. The factor behind the less percenatge of code coverage of our approach in comaprision to MFTS[34] is the that each test suffers with a significant % of reduction in the test suite size. But, the other important caharacteristics of our approach in comparision with the esisting approahes [6,33,34] is that we have not discarded the redundant and the obsolete test cases perminantely from the test suites instead of that we preserve them into their clusters. Therefore it become possible to increase the % of code coverage by considering a different population of test cases from each clustered test suites using our approach. Hence, our approach is more flexibile then the existing approaches [6, 33, 34].

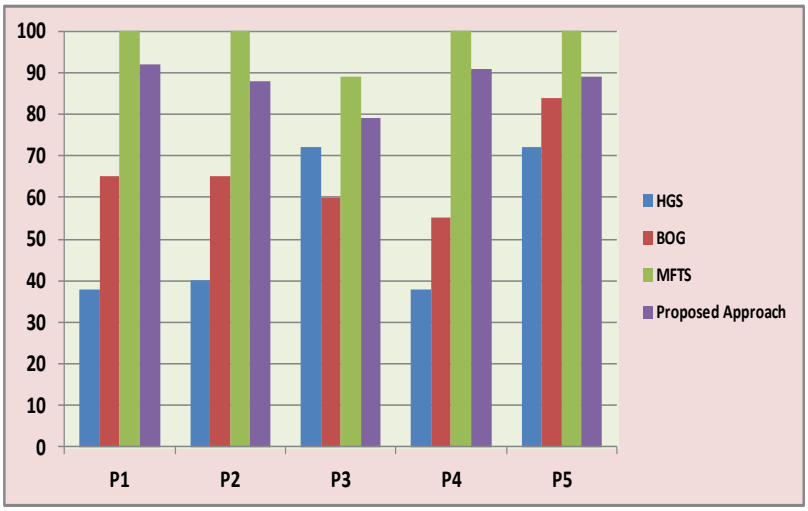

Fig. 16. Code Coverage Comparison with other Approaches

The other important quality attribute that is used to evaluate the quality of the minimized test suites is the percentage of fault detection loss. The % of fault detection loss is a measure that indicates the % of fault detection loss occurred to the minimized test suite with respect to the original test suite. It is calculated using the below mentioned equation:

$$\% \text{ of Fault Detection Loss} = \frac{F_{red} - F}{F} \times 100$$

Where |F| deneotes the total number of faults identified by the original test suite before minimization and $|F_{red}|$ is number of faults idntified by the minimized test suite. Although we have not calculated the fault detection loss of the minimized test suites but it is assumed to be minimum as we have not dicarded the test cases from the test suites. Intially, if the fault dection loss of the intial selection or seed of test cases from each clustered test suites can be higher due to the higher percentage of reduction in the size of the test suite, but it can also be reduced by considering another population or seed of test cases from the each clustered test suites. The percentage of reduction in the size will still remain significant as shown in the table 4.

This is one of the good chararcteristics of the proposed approach that it does not throw away test cases that are redundant as compared to other approaches but groups them in different clusters and then considers only a subset from the whole test suite for testing the code. So, if the fault detection effectiveness is found to be degrading it can also be improved by considering one more subset or population of test cases from each clustered test suites for re-testing the code.

Table- 4. Percentage of Reduction after Considering one more Subset from the Clusterd Test suites.

Subject Programs	Initial Test suite Size (T)	Size after Minimization (T_{rs}) with Another Subset of Test Cases	% of size Reduction
P1	113	8	**92.9%**
P2	100	10	**90%**
P3	100	10	**90%**
P4	100	4	**96%**
P5	35	6	**82.8%**

V. CONCLUSION AND FUTURE SCOPE

Software testing is very important and challenging activity. Testing techniques should find the possible number of faults or errors within a limited time and with a finite number of test cases. But automatic software test case design or generation algorithms are exhaustive with respect to the coverage goal defined. Therefore, if a coverage criterion is not properly chosen, they may generate too many test cases that are infeasible to be considered for practical execution. Also, most of the test cases in the automated generated test suite can be redundant in the sense of exercising common features of the code under test and revealing common sets of defects. Therefore, other than structural coverage criteria, test-case generation may need to be combined with an effective set of selection strategies that will minimize redundancy in test suites; and limit the size of test suites. For the same goal, we in this study propose a hierarchical clustering approach for the minimization of size and redundancy in an automated generated test suite. With the proposed approach, we first segmented the initial automated test suites into a fixed number of partitions or clusters based on the value of minimum requirement coverage criteria. The minimum requirement coverage is the basic percentage of code coverage that at least must be achieved by the minimized test suites. After clustering with the proposed technique, we selected a single test case from cluster in each clustered test suites in order compose new test suites that we call the minimized test suite (MTS) for the execution. The minimized test suites (MTS's) after selection are evaluated based on the two well known software testing metrics. The first metric known as percentage of reduction in the test suite size gained through the proposed approach and the second important metric is the percentage of code coverage achieved by the minimized test suites. It is observed through our experimentation that a significant amount of percentage reduction in the size of each test suite is resulted with the proposed approach. Also through our experimentation an acceptable level of code coverage is also achieved by the minimized test suites for each subject program. It was also observed that our approach compared to other approaches does not throw away redundant test cases from the test suites but groups them according to the requirements they satisfy. Hence, the fault detection effectiveness of the test suite is preserved as we consider at any instant only a single representative from each cluster for execution. The limitations of the proposed approach is that, it at any instant results in a minimized test suite that is adequate with respect to minimum requirement coverage criterion, but if the adequacy of the minimized test suite is not of acceptable level, we may need to consider more test cases in order to improve the adequacy. The future scope of this study will be the application of more appropriate minimum requirement coverage criteria for test suites minimization and its evaluation with respect to other clustering algorithms.

REFERENCES

1. A. Smith, J. Geiger, M. Kapfhammer and M. Soffa, "Test suite reduction and prioritization with call trees", in Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering (ASE '07), 2007, pp. 539-540.
2. S. Parsa, A. Khalilian, "On the optimization approach towards test suite minimization", International Journal of Software Engineering and its applications, January 2010, Vol. 4, No. 1.
3. S. Selvakumar, N. Ramaraj, "Regression test suite minimization using dynamic interaction patterns with improved FDE", *European Journal of Scientific Research*, 2011, Vol. 49, No. 3, pp. 332-353.
4. H. Hsu, A. Orso, "MINTS: A general framework and tool for supporting test suite minimization", International Conference on Software Engineering (ICSE 09), 2009.
5. R .V. Binder, "Testing Object-Oriented Systems Models, Patterns, and Tools, Object Technology Series", 1999, Addison Wesley, Reading, Massachusetts, October.
6. M. J. Harrold, R. Gupta, M L Soffa, "A methodology for controlling the size of a test suite", ACM Transactions on Software Engineering and Methodology, 1993, vol. 2 (3), pp. 270–285.
7. J. A. Jones, M. J. Harrold, "Test-Suite Reduction and Prioritization for Modified Condition/Decision Coverage" IEEE Transactions on Software Engineering, March 2003, vol. 29(3), pp.195-209.
8. H. Agrawal, "Dominators, Super Blocks, and Program Coverage", 21st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Portland, Oregon, January 1994, pp. 25-34.
9. G. Rothermel, M. J. Harrold, J Ostrin, and C Hong, "An Empirical Study of the Effects of Minimization on the Fault Detection Capabilities of Test Suites." International Conference of Software Maintenance, Bethesda, Maryland, November 1998, pp. 34-43,
10. W. E. Wong, J. R. Horgan, S. London, and A. P. Mathur, "Effect of Test Set Minimization on the Fault Detection Effectiveness of the All-Uses Criterion." Technical Report SERC-TR-152-P, Software Engineering Research Center, June 1994.
11. W. E. Wong, J. R. Horgan, S. London, and A. P. Mathur, "Effect of Test Set Minimization on Fault Detection Effectiveness", Proc. 17th Int'l Conference on Software Engineering, , Washington, Seattle, April 1995, pp. 41-50.
12. W. E. Wong, J. R. Horgan, S. London, and A. P. Mathur, "Effect of Test Set Minimization on Fault Detection Effectiveness." Software Practice and Experience, April 1998, vol. 28(4), pp.347-369.
13. D. Jeffrey, N. Gupta", Test suite reduction with selective redundancy", In Proceedings of the 21st IEEE International Conference on Software Maintenance, Washington, DC, USA, IEEE Computer Society, 2005, pages 549–558.
14. S.Tallam and N. Gupta, "A concept analysis inspired greedy algorithm for test suite minimization", In Proceedings of the 6th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering (PASTE '05), 2005, pp. 35-42.
15. Z. Chen, B. Xu, X. Zhang and C. Nie, "A novel approach for test suite reduction based on requirement relation contraction", in Proceedings of the 2008 ACM symposium on Applied computing (SAC '08), 2008, pp. 390-394.
16. N. Yevtushenko, A. Cavalli, and R. Anido, "Test Suite Minimization for Embedded Nondeterministic Finite State

Machines", in Proceedings of the IFIP TC6 12th International Workshop on Testing Communicating Systems: Method and Applications, 1999, pp. 237-250.
17. M. Usaola, P. Mateo, and B. Lamancha, "Reduction of test suites using mutation", In Proceedings of the 15th international conference on Fundamental Approaches to Software Engineering (FASE'12), 2012, pp. 425-438.
18. N. Mansour, K. El-Fakih, "Simulated annealing and genetic algorithms for optimal regression testing", Journal of Software Maintenance, 1999, vol. 11 (1), pp. 19–34.
19. A. Bertolino, E. Marchetti, "A Brief Essay on Software Testing", Technical Report: TR-36, 2004.
20. A. Bertolino, "Software testing research: achievements, challenges, dreams", In Proceedings of the 1st Workshop on Future of Software Engineering (FOSE'07), at ICSE, 2007, pp 85–103.
21. M. Pezzè, M. Young, "Software Testing and Analysis, Process, Principles and, Techniques", Wiley 2007.
22. H. Zhu, P. A. V Hall, J. H. R. May, "Software unit test coverage and adequacy", ACM Computing Surveys, 1997, vol. 29 (4), pp. 366–427.
23. K. P. Chan, T. Y. Chen and D. Towey, "Normalized restricted random testing", In Reliable Software Technologies Ada-Europe, 2003, pages 368–381, Springer.
24. J. C. King, "A new approach to program testing", In Programming Methodology (LNCS), 1975, Vol. 23, pp. 278-290.
25. W. Miller and D. Spooner, "Automatic Generation of Floating-Point Test Data", IEEE Transactions on Software Engineering, 1976, vol. 2, no. 3, pp. 223-226,
26. A. K. Jain, "Data clustering: 50 years beyond k-means", Pattern Recognition Letters, 2010, vol. 31(8), pp. 651– 666.
27. D. J. Bora, A. K. Gupta, "A Comparative study Between Fuzzy Clustering Algorithm and Hard clustering Algorithm", International Journal of Computer Trends and Technology (IJCTT) , Apr 2014,vol. 10 no. 2, pp. 108-113.
28. F. A. Khan, A. K. Gupta, D. J. Bora, "Profiling of Test Cases with Clustering Methodology", International Journal of Computer Applications 106(14):32-37.
29. R. C. Dubes, A. K. Jain, "Algorithms for Clustering Data, Prentice Hall, 1988.
30. L. Kaufman, P. J. Rousseeuw, "Finding Groups in Data: an Introduction to Cluster Analysis", (1990), John Wiley and Sons.
31. The random test data generation tool, http://www.generatedata.com/.
32. D. S. Johnson, "Approximation algorithms for combinatorial problems", In Journal of Computer and System Sciences, 1974, vol. 9 (3), pp. 256–278.
33. P. Saeed, and A. Khalilian. "On the optimization approach towards test suite minimization" , International Journal of Software Engineering and its applications 4.1 (2010): 15-28.
34. P. Harris, N. Raju, " Towards test suite reduction using maximal frequent data mining concept", International Journal of Computer Applications in Technology, 2015, Vol. 52(1), pp.48-58.

AUTHORS PROFILE

Dr. Fayaz Ahmad Khan has completed his Ph.D. in June 2018 form department of computer science & applications Barkatullah University Bhopal. Dr. Khan is currently engaged in Teaching and research in the same department. His research areas include Software Engineering, Software Testing and Data Mining.