doi: 10.21917/ijsc.2013.0094

FPGA BASED SOFTWARE TESTING PRIORITIZATION USING RnK-MEANS CLUSTERING

N. Bharathi[1] and P. Neelamegam[2]
[1]*School of Computing, SASTRA University, India*
E-mail: bharathi_n@cse.sastra.edu
[2]*School of Electrical and Electronics Engineering, SASTRA University, India*
E-mail: neelkeer@eie.sastra.edu

Abstract

Testing the software is to validate its correctness when it is deployed in its actual environment. Various test cases should be implemented and tested to validate the software. When more than one test case is involved, the order of testing needs to be prioritized to optimize the testing process. This paper proposed a prioritization method with repeated n times K means (RnK-means) clustering. Priority for the test cases is assigned based on the cluster mean values by executing RnK-means for each factor of test cases. Existing techniques are calculating merely the average of factor weights for each test case for deciding priority. The proposed method involves K-means computations and it is accelerated by FPGA for deciding priority. The observed results proved 20 percent better performance with RnK-means clustering than the existing weighted average method.

Keywords:

K-means Clustering, Field Programmable Gate Array (FPGA), ATM Application, Scalability, User Friendliness

1. INTRODUCTION

Any computing system comprises of hardware as well as software to achieve a task. Hardware is all-inclusive of physical components and wires of the computing system. Certification of correctness is checked at the manufacturer site itself and its damage or fault can be checked easily by viewing and connecting it together using wires. Software fault cannot be detected easily by simply viewing it [1].

At present, releasing of new software components and applications are rapidly increased. The important and most asked question is which software is better that satisfies the various requirements of the end users and what certifies that this software is correct and gives reliable outcome. The answer to this question is software testing. Testing is verifying whether the software is satisfying the necessary conditions stated at the commencement of its development phase and validating whether it meets the specified user requirements [2]. There are different levels of testing in which test cases are applied to software. Unit testing verifies single block or portion of code e.g. subroutine/function etc. Integration testing verifies the interfacing between the components or block of codes, communications between the subroutines and functions. System testing tests the complete set of codes in its final form [3]. Testing can be done in two ways. The software developers themselves test their own software for its validation during development life cycle [4]. Other approach is deputing a team for testing the software. The software testing team should design various test cases to validate and verify the correctness of the software.

Test cases are the commonly used method for testing the software [3]. Gamut of test cases from more general case that can be applied to any software, to more specific case applied to particular software are available for testing. When new software is developed, some set of general test cases should be executed always to verify its basic requirements [5]. Those test cases are assigned with highest priority and only after passes through it, the software is subjected to others. If there are N test cases, then N! possible orders exist with which the test cases can be applied on the software for testing it [6]. The order is enforced by assigning certain priorities for the test cases based on some factors that are influencing the correctness of the software or detecting the faults. Various prioritization techniques exist to assign priority to test cases [7]. Based on the priority order the test cases are applied on to the software and test log is also maintained. The priority order in applying the test cases is to ensure the quality of software. Besides it detects the possible faults in the program and failures for certain set of inputs. On the whole it should state the effectiveness of the software in satisfying the necessary end user requirements.

A prioritization technique should give importance to the weights assigned to the attributes of the test cases [8]. It should consider the factors that are highly influencing the detection of faults. The existing methods selects the best one from the list of test cases ranked based on the average weights. But if more than one factors involved then choosing the test case with maximum average weight is not always providing better results. Even though it is maximum value it may be lagging in any one of the factor taken which is contributing more to the testing. Hence this work proposed an efficient RnK means clustering algorithm which is based on k-means clustering. Instead of simply selecting the maximum average weight, it follows a novel heuristic in selecting the priority of the test cases.

K-means clustering is a method of categorizing or partitioning the given set of elements into disjoint groups [9]. It generates flat non overlapping clusters which is highly suitable for assigning priority to software test cases. Elements that have similar property are grouped under the same category and elements belong to the different category are different in nature. The k-means algorithm is as follows with a data set S Real number and an integer k:

```
Initialize centers $z_1, \ldots, z_k \in$ Real number and clusters $C_1, \ldots, C_k$ in any way
repeat until there is no further change in cost:
for each $j$: $C_j \leftarrow \{x \in S$ whose closest center is $z_j\}$
for each $j$: $z_j \leftarrow \text{mean}(C_j)$
```

This is simple to implement, and takes $O(k|S|)$ time per iteration. It is implemented in FPGA to speed up the process [10].

FPGA is selected for its rich characteristics like architectural adaptability, enhanced speed, comfortable routing, and new, refined hardware definition language to realize completely automatic implementation of composite, large, high performance circuits [11]. FPGAs require no assessment vectors to generate and no bottleneck while waiting for archetype to be manufactured as they are software oriented and consumer programmed [12]. Changes of configuration bits are taken place on the fly and safe. Circuits can be realized in the order of minutes unlike ordinary gate arrays which take weeks of time. FPGA devices made a revolution with significant cost reduction in circuit design and fabrication. FPGA devices possess matrix like architecture with logic cells enclosed by a frame of I/O cells. This paper first explains the prioritization of software testing and designing of clustering algorithm with FPGA followed by the architectural framework of the proposed method with its implementation. Finally, the proven results are discussed with the ATM software application [13].

2. SOFTWARE TESTING PRIORITIZATION

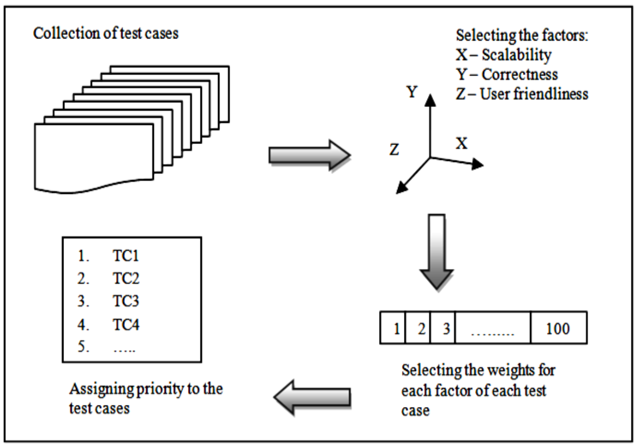

Fig.1. Prioritization of test cases

A collection of test cases is designed for newly developed software besides the existing general test cases. The objective is to find the order of applying the test cases to facilitate the detection of faults rapidly using efficient technique. To enforce the order, priority for the test cases is assigned and tested on the software based on it. Priority is assigned based on the factors that are possessed by the test cases to reveal the quality of the software as illustrated in Fig.1. The selection of factors should be in such a way that it covers all the criterion of software requirements. The selected factors are scalability, Correctness and User friendliness. After selecting the factors, the weights for each factor of each test case should be determined to reflect the capability of the test case to verify and validate the software. Based on the weights, the priority is assigned for each test case. The test cases are then applied on the software to check the efficiency based on the priority assigned. Hence the prioritization method has the main control in deciding how better the software is.

3. FPGA BASED CLUSTERING

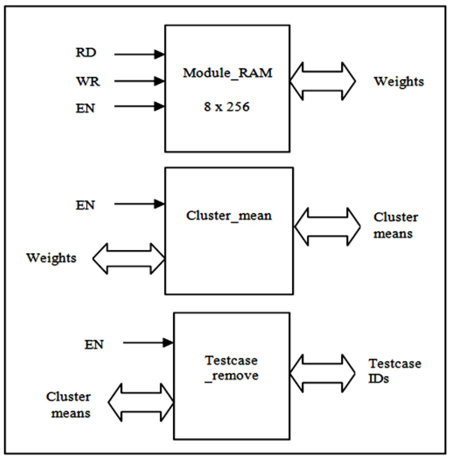

Fig.2 Hardware modules of RnK-means clustering

Clustering is performed in FPGA by designing the hardware circuit for clustering algorithm. The modules are memory_ram module to store the weights, the cluster_mean module to calculate the cluster means, Testcase_remove module to remove the selected test cases in every execution of k-means clustering algorithm. The input and output signals of the modules are shown in Fig.2. The modules are implemented and simulation results are checked and verified with the results obtained in traditional C implementation. The benefit of using FPGA to configure the clustering algorithm as hardware circuit is that the circuit can be modified as and when needed for different sizes of data. It can also be tailored to meet the requirements of various applications that use clustering as a classification algorithm.

4. ARCHITECTURE FRAMEWORK

The layered architecture of the proposed method is demonstrated in Fig.3. The software is developed and subjected to testing to evaluate its efficiency and effectiveness as it is deployed in real destined environment. Test cases are designed in such a way to validate the correctness of the software by simulating the real environment where the software is to be deployed. Table.1 illustrates the different fields of the test cases. Among the various test cases implemented, selecting the appropriate test cases which reveals more no. of faults in lesser time in the software is more significant. The order of executing the test cases contributes more in revealing the faults rapidly. Attributes of the test cases which are detecting the faults are the deciding factors of the order of applying the test cases. Hence priority is assigned based on those factors possessed by the test cases. Each factor of the test cases is assigned with a weight ranges from 1 to 100 based on how much efficient that test case is in validating the software.

Table.1. Fields and its definition of a Test case

Field	Definition
Test case ID	The unique identity of the test case
Test priority	Highest, high, medium, low,

	lowest – used for prioritization
Module Name	Name of the main module
Test Designed By	Name of the tester who designed it
Test Designed Date	Date when it is written
Test Executed By	Name of the tester who executed this test case
Test Execution Date	Date on which test is executed
Test Title/Name	Test case name
Test Summary/Description	Purpose of the test case
Pre-condition	State before the execution of test
Dependencies	Dependent with any other test case
Test Steps	Order of test steps etc.
Test Data	Input given at the time of testing
Expected Result	Predictable output
Post-condition	State after the execution of test
Actual result	Actual output after applying the test
Status (Pass/Fail)	Checking the equality of expected and actual result and specify success if equal
Notes/Comments/Questions	Any special or exceptional cases

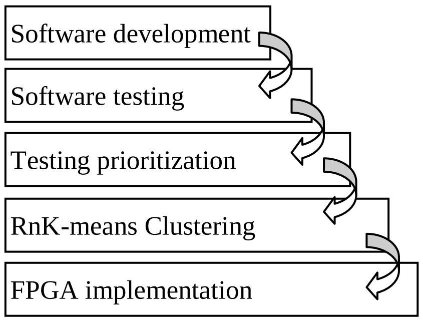

Fig.3. Layered architecture

The test case is assigned with the weight that reflects the effectiveness of software in that dimension. Three factors are selected to efficiently validate the software. Scalability which checks the whether the software cope with the increase in the number of users/memory space etc. Correctness is for checking the software works well under any circumstances and for any user input without aborting the execution. User friendliness is one of the important factors because, though the software is performing extraordinarily well, if it is not providing user friendliness in operating with, then it becomes inefficient.

The factor weights of the test cases are categorized into any one of 5 clusters using RnK means clustering algorithm. The clusters are highest, high, medium, low and lowest. The test cases of highest cluster are assigned with the highest priorities followed by high, medium, low and lowest cluster test cases. The additional time spent in pre-computation for RnK means clustering is speed up by the FPGA implementation of the algorithm.

5. IMPLEMENTATION

The set of test case is designed and weights ranging between 1 and 100 are assigned according to their ability to validate the software and detecting the faults. Scalability, correctness and user friendliness are the three factors considered for each test cases. The average weights of these three factors for each test case are pre-calculated. With these weights the test cases are subjected to RnK means algorithm. The algorithm assigns priority to the test cases as follows: Each test case is assigned with three weights one for each factor. Weights of each test case for scalability are subjected to k-means clustering with initial cluster centers as 100, 80, 60, 40 and 20. The clusters are highest, high, medium, low and lowest. This procedure is repeated for correctness and user friendliness with their respective weights of test cases.

After completing the execution, the highest cluster mean is taken for all the three factors and its average (AHT1) is calculated. The closest pre-calculated average weight of test case to the AHT1 is determined and that test case is removed from the test case set. This procedure is repeated for finding average of high cluster means AH1, medium cluster means AM1, low cluster means AL1 and lowest cluster means ALT1. The closest pre-calculated average weight of test case to AH1, AM1, AL1 and ALT1 are also determined and its respective test cases are removed. The removed test cases are categorized under priority groups of highest, high, medium, low and lowest. The priority groups for the remaining test cases are determined by repeated executions of k-means. AHT2 …AHTn1, AH2…AHn2, AM2…AMn3, AL2…ALn4 and ALT2…ALTn5 are determined in the repeated executions where n1, n2, n3, n4 and n5 are the no. of test cases under highest, high, medium, low and lowest priority groups respectively.

The priority of test cases with in a group is assigned in the order in which the test cases are removed. Early removal gets higher priority. The priority across the groups is ordered as first priority of the high group is the next to the last priority of the highest group. Similarly first priority of the medium group is the next to the last priority of the high group, first priority of the low group is the next to the last priority of the medium group, first priority of the lowest group is the next to the last priority of the low group.

Factors:

Scalability – supports to any no. of users

Correctness – satisfying programming view and user requirement view.

User friendliness – should be easily usable by the customers

Clusters for the test case:

Highest: Exactly matching the software requirements.

High: Matching with software but slight deviation with requirements.

Medium: Deviating slightly with software and matching with requirements.

Low: Deviating slightly with software as well as its requirements.

Lowest: Different from software and its requirements

Algorithm: RnK-means

```
Step1: Assign weights for 1 to 3 factors between 1 and 100
       based on software requirement.
Step 2: Repeat step 4 for each factor $i$ = 1 to 3.
Step 3: Weight of factor i of all test cases is subjected to K-
       means clustering.
Step 4: K-means is performed by taking five clusters as highest,
       high, medium, low and lowest.
Step 5: Take the highest cluster mean (HTCM) for each factor
       and find the average (AHT1).
Step 6: Do step 6 for high, medium, low and lowest and find the
       averages (AH1, AM1, AL1 and ALT1).
Step 7: Scan for the closest match test case (TCHT1, TCH1,
       TCM1, TCL1, TCLT1) of AHT1, AH1, AM1, AL1 and
       ALT1 respectively in the weighted average ranking.
Step 8: Assign the highest priority of highest group to TCHT1
       and remove it from the test case set.
Step 9: Do step 9 for TCH1, TCM1, TCL1 and TCLT1.
Step 10: Repeat the procedure from step 2 to step 10 till no test
       cases for assigning priority.
Step 11: Rank test cases with highest group first followed by
       high, medium, low and lowest groups in order.
```

The test cases TCHT1, TCH1, TCM1, TCL1 and TCLT1 are determined as follows:

{HTCM, HCM, MCM, LCM, LTCM} of scalability = k-means (scalability weights of 50 test cases)

{HTCM, HCM, MCM, LCM, LTCM} of correctness = k-means (correctness weights of 50 test cases)

{HTCM, HCM, MCM, LCM, LTCM} of user friendliness = k-means (user friendliness weights of 50 test cases)

AHT1 = (HTCM of scalability + HTCM of Correctness + HTCM of User friendliness) / 3

$$\text{TCHT1} = \text{closestmatch(AHT1)} \quad (1)$$

AH1 = (HCM of scalability + HCM of Correctness + HCM of User friendliness) / 3

$$\text{TCH1} = \text{closestmatch(AH1)} \quad (2)$$

AM1 = (MCM of scalability + MCM of Correctness + MCM of User friendliness) / 3

$$\text{TCM1} = \text{closestmatch(AM1)} \quad (3)$$

AL1 = (LCM of scalability + LCM of Correctness + LCM of User friendliness) / 3

$$\text{TCL1} = \text{closestmatch(AL1)} \quad (4)$$

ALT1 = (LTCM of scalability + LTCM of Correctness + LTCM of User friendliness) / 3

$$\text{TCLT1} = \text{closestmatch(ALT1)} \quad (5)$$

Similarly TCHT2, TCH2, TCM2, TCL2 and TCLT2 and so on up to TCHTn1, TCHn2, TCMn3, TCLn4 and TCLTn5 are determined by repeated execution of K-means. In the above equations, certain cluster means may be zeros based on the weights of testcases and clustering.

6. RESULTS AND DISCUSSION

The test cases are designed and implemented. The priority of the test cases is determined based on the proposed method. The results of the proposed method are proving the accuracy of the priority assigned to the test cases by validating the software efficiently and detecting the faults rapidly. A set of 50 test cases are designed and tested for an ATM application. The test cases are as follows:

TC 1:	successful card insertion.
TC 2:	unsuccessful operation due to wrong angle card insertion.
TC 3:	check for displaying the message for proper insertion of card.
TC 4:	unsuccessful operation due to invalid account card (not at all a debit/credit/cash card).
TC 5:	unsuccessful operation due to broken card.
TC 6:	unsuccessful operation due to expired card.
TC 7:	check for displaying the message for invalid card.
TC 8:	check for displaying the message for broken card.
TC 9:	check for displaying the message for expired card.
TC 10:	check for other banks card.
TC 11:	check for displaying the message for other bank card with service charge.
TC 12:	successful entry of pin number.
TC 13:	unsuccessful operation due to lesser number of character in pin.
TC 14:	unsuccessful operation due to wrong pin number entered 1 time.
TC 15:	unsuccessful operation due to wrong pin number entered 2 times.
TC 16:	check for displaying the message for entering wrong pin number.
TC 17:	unsuccessful operation due to wrong pin number entered 3 times.
TC 18:	check for displaying the message for blocking the card.
TC 19:	successful selection of language.
TC 20:	successful selection of account type.
TC 21:	unsuccessful operation due to wrong account type selected w/r to that inserted card.
TC 22:	check for displaying the message for wrong account type w/r to the inserted card.
TC 23:	check for displaying the available options (balance enquiry, withdrawal etc)
TC 24:	successful selection of withdrawal option.
TC 25:	successful selection of amount.
TC 26:	unsuccessful operation due to wrong denominations.

TC 27:	check for displaying the message for wrong denominations.
TC 28:	successful withdrawal operation.
TC 29:	check for displaying the message for taking cash from the ATM.
TC 30:	unsuccessful withdrawal operation due to amount greater than possible balance.
TC 31:	check for message display for lack of requesting amount in the account.
TC 32:	unsuccessful due to lack of amount in ATM.
TC 33:	check for message display for lack of amount in ATM.
TC 34:	unsuccessful withdrawal operation due to not able to print receipt
TC 35:	unsuccessful withdrawal operation due to not taking cash from ATM in specified time.
TC 36:	successful selection of receipt printing.
TC 37:	unsuccessful selection of receipt printing.
TC 38:	undue to amount greater than the day limit.
TC 39:	undue to server down.
TC 40:	undue to ATM out of order.
TC 41:	undue to click cancel after insert card.
TC 42:	undue to click cancel after insert card and pin no.
TC 43:	undue to click cancel after language selection.
TC 44:	undue to click cancel after account type selection.
TC 45:	undue to click cancel after withdrawal selection.
TC 46:	undue to click cancel after entering amount.
TC 47:	successful selection (to continue) of next transaction.
TC 48:	successful selection of balance enquiry option.
TC 49:	check for displaying balance amount in the account.
TC 50:	successful selection of exit.

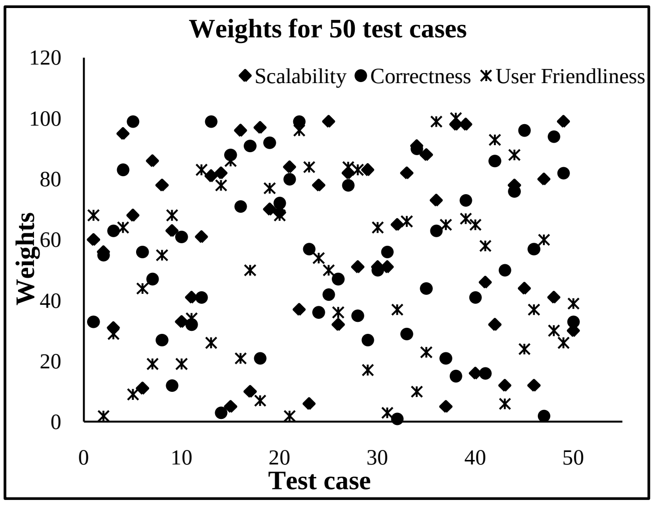

Fig.4(a).

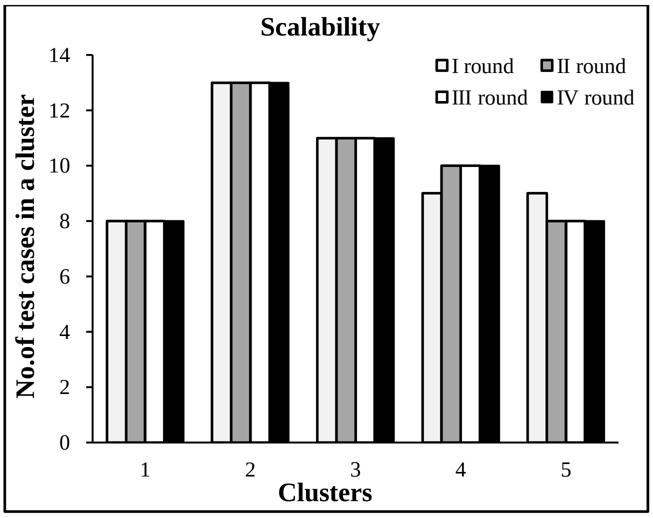

Fig.4(b).

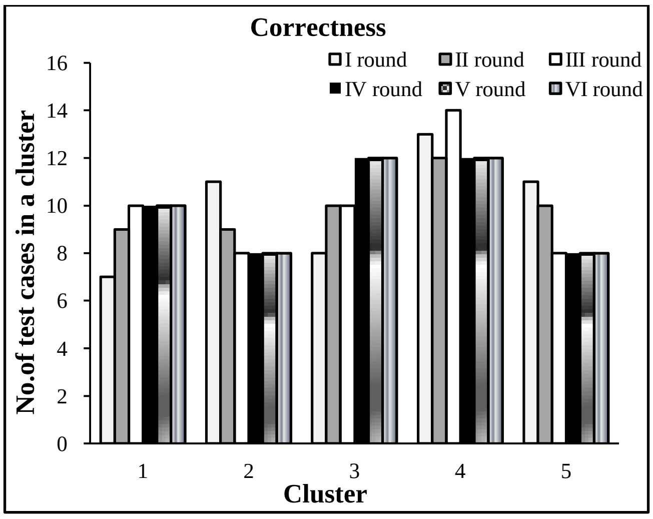

Fig.4(c).

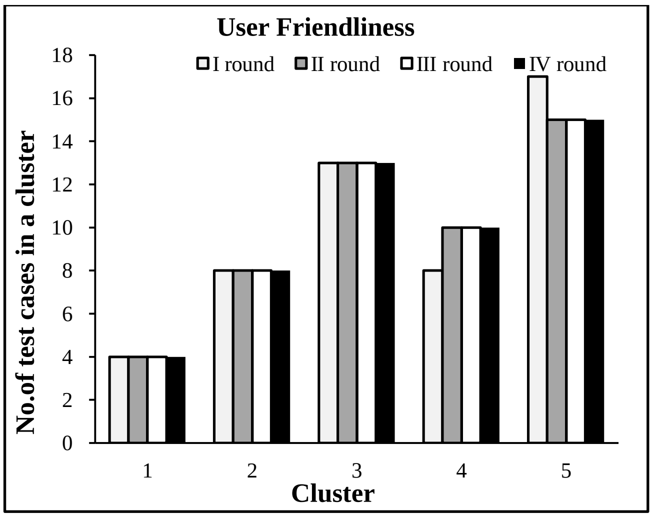

Fig.4(d).

Fig.4(a). Three factor weights of 50 test cases (4(b), 4(c) and 4(d)) HTCM, HCM, MCM, LCM and LTCM of scalability, correctness and user friendliness respectively

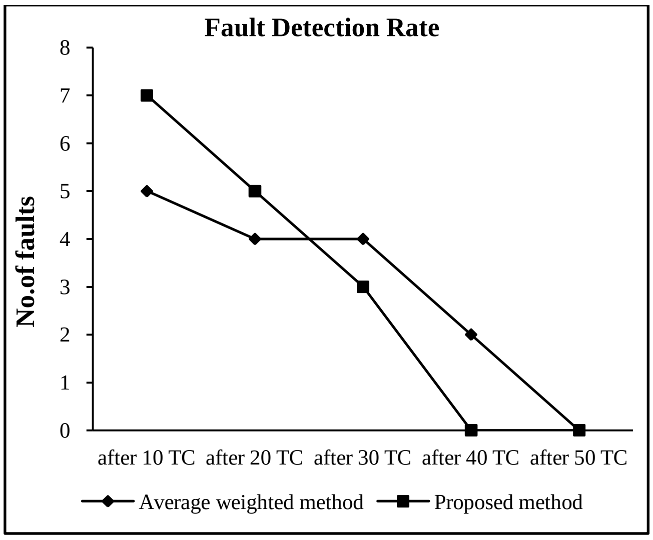

Fig.5. Comparative graph of fault detection rate

The Fig.4(a) shows the scalability, correctness and user friendliness weights of the 50 test cases that range from 1 to 100. The Fig.4(b) to Fig.4(d) shows the five cluster means after first execution of k-means algorithm. The software faults are intentionally created and tested with the test cases. The Fig.5 proves the better fault detection rate of proposed method over average weighted method. Proposed method detects 12 faults with the total of 15 faults with first 20 testcases whereas the average weighted method detects only 9 faults.

7. CONCLUSION

The proposed test cases prioritization mechanism is implemented for testing the software effectively based on clustering. FPGA is used to fasten the process of assigning priority to the test cases in order to verify the software and detect the faults rapidly. 50 test cases are taken and are categorized in the scale of clusters as highest, high, medium, low and lowest. The highest category is assigned with highest priority and the successive priorities are assigned with next immediate categories till lowest. With the resultant priority for the test cases it is proved that software is tested efficiently and the fault detection rate is improved nearly 20 percent. The increased number of test cases is necessary to get better fault detection rate. This work also concludes that test cases categorized under highest and high should necessarily be applied on to validate the software.

REFERENCES

[1] Hayden C. M, Smith E. K, Hardisty E. A, Hicks M and Foster J. S, “Evaluating dynamic software update safety using systematic testing”, *IEEE Transactions on Software Engineering*, Vol. 38, No. 6, pp. 1340-1354, 2012.

[2] Kuhn D. R, Wallace D. R and Gallo A. M. Jr., “Software fault interactions and implications for software testing”, *IEEE Transactions on Software Engineering*, Vol. 30, No. 6, pp. 418-421, 2004.

[3] Bertolino and Antonia, “Software Testing Research: Achievements, Challenges, Dreams”, *Proceedings of the Future of Software Engineering*, pp. 85-103, 2007.

[4] Mustafa K M, Al-Qutaish R. E, Muhairat M. I, “Classification of Software Testing Tools Based on the Software Testing Methods”, *Proceedings of Second International Conference on Computer and Electrical Engineering*, Vol. 1, pp. 229-233, 2009.

[5] Datchayani M, Arockia Xavier Annie R, Yogesh P and Zacharias B, “Test case generation and reusing test cases for GUI designed with HTML”, *Journal of Software*, Vol. 7, No. 10, pp. 2269-2277, 2012.

[6] Gend Lal Prajapati, “On the Inference of Automatic Generation of Software Tests”, *Proceedings of Fourth International Conference on Emerging Trends in Engineering & Technology*, pp. 18-21, 2011.

[7] Mei H, Hao D, Zhang L, Zhang L, Zhou J and Rothermel G, “A static approach to prioritizing JUnit test cases”, *IEEE Transactions on Software Engineering*, Vol. 38, No. 6, pp. 1258-1275, 2012.

[8] Masayuki Hirayama, Osamu Mizuno and Tohru Kikuno, “Test Item Prioritizing Metrics for Selective Software Testing”, *IEICE Transactions on Information and Systems*, Vol. E86–D, No. 3, pp. 2733-2743, 2004.

[9] Christ M. C. J and Parvathi R. M. S, “Segmentation of Medical Image using K-Means Clustering and Marker Controlled Watershed Algorithm”, *American Journal of Applied Sciences*, Vol. 8, No. 12, pp. 1349-1352.

[10] An F, Koide T and Mattausch H. J, “A K-means-based multi-prototype high-speed learning system with FPGA-implemented coprocessor for 1-NN searching”, *IEICE Transactions on Information and Systems*, Vol. E95-D, No. 9, pp. 2327-2338, 2012.

[11] Kumar A and Anis M, “IR-drop aware clustering technique for robust power grid in FPGAs”, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 19, No. 7, pp. 1181-1191, 2011.

[12] Bharathi N and Neelamegam P, “FPGA based linear heating system for measurement of Thermoluminescence”, *Measurement Science Review*, Vol. 11, No. 6, pp. 207-209, 2011.

[13] Luo H, Masud M and Ural H, “Detecting offline transaction concurrency problems”, *Journal of Software*, Vol. 7, No. 8, pp. 1855-1860, 2012.