IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Vol.-7, Issue-6, June 2019 E-ISSN: 2347-2693

# Package Level Test Case Minimization for Bug Prediction using Linear Regression Machine Learning Approach

**Divya Taneja[1*], Rajvir Singh[2], Ajmer Singh[3]**

[1,2,3] Department of Computer Science and Engineering, Deenbandhu Chhotu Ram University of Science and Technology, India

*Corresponding Author:divya.taneja20.dt@gmail.com, Tel.: 8295610675

DOI: https://doi.org/10.26438/ijcse/v7i6.364370 | Available online at: www.ijcseonline.org

Accepted: 11/Jun/2019, Published: 30/Jun/2019

***Abstract*—** With the growing complexities in Object Oriented (OO) software, the number of bugs present in the software module is increased. In this paper, a technique has been presented for minimization of test cases for the OO systems. The Camel 1.6.1 open source software was used the evaluation of proposed technique. The mathematical model used in the proposed methodology was generated using the open source software WEKA by selecting effective Object Oriented (OO) metrics. Ineffective and effective Object Oriented metrics were recognized by using the techniques based on feature selection to generate test cases that cover fault proneness classes of the software. The defined methodology used only effective metrics for assigning weights to test paths for minimization. The results show the significant improvements.

***Keywords****:* Camel 1.6.1, Test Case Minimization, WEKA

## I. INTRODUCTION

Presence of defects in software modules reduces the quality of the software. It is necessary to improve the quality of software by identifying defects and remove them from the software module to deliver reliable software product[1].In testing phase s development team finds the defects present in the software module . It is costly to test the entire software within a limited period of time. As a result, less reliable and defective software is released. It becomes necessary to eliminate software defects within limited time and less cost. Software defect prediction model is one of the solutions to this problem. The use of software defect prediction model is to predict the defects present in the software. To select the minimum number of test cases test case minimization technique is used which are able to reveal more software defects within less time and cost.

The software defect prediction model is skilled by using attributes of the software and fault data according to the previously released software or identical projects. This information is used to calculate whether the software module is defective or not. The effectiveness and performance of software defect prediction model rely on the characteristics of the software attributes that are used to calculate or predict whether defects are present in the software module or not.

### A. *Organization of this paper*

The organization of paper is as follows. In Section II we provide information about the background or related work on software defect prediction using software metrics. In Section III provide details about the methodology used in this paper. In Section IV implementation steps are defined. Section V shows the results. Section VI and section VII shows conclusions and future work respectively.

## II. RELATED WORK

In 2016, S. Puranik, P. Deshpande, and K. Chandrasekaran [2] researchers select only the minimum number of software metrics. The researchers proposed an algorithm that predicts the bug proneness index by using marginal R-square values method. The regression testing was perform as mediator step in this given algorithm, it was found that they was different in nature when they compare with the models by using regressions alone.

In 2013, S. Prateek, A. Pasala, and L. M. Aracena [3] performed an analysis to check the effectiveness of the network metrics above code metrics for the prediction of bug. This work was carried out on the 11 datasets from the Open source PROMISE repository [4] by using three different machine learning algorithms. For each project the binary classification model was built to identify the files that contain bugs in different condition. It was seen that the code metrics was predict better results over the network metrics.

In 2018, A. Singh, R. Bhatia, and A. Singhrova [5] discussed the usage of Object Oriented metrics for fault prediction. The effective metrics was used to determine the quality of the software module. It was concluded that the code metrics needed in minimizing the efforts required in software during the maintenance phase of the software.

In 2018, M. Akour and L. Abuwardih [6] found that a large number of test suite were required to test the software module and different methods were required to reduce the test cases. The study was conduct to deal with the effectiveness of genetic algorithm (GA) in order to reduce the number of test cases. The GA steps were repeated to minimize test suite.

In 2017, D. L. A. L. Gupta and K. Saxena [7] proposed software bug prediction system (SBPS) . The model predicted the bugs present in a class using metrics. The model forecasted the occurrence of bugs in a class during test of the software. They formulate hypotheses corresponding to each metric. The logistic regression classifier gives high accuracy among all classifiers used in this study.

In 2005, R. Ferenc [8] described how to calculate the Chidamber and Kemerer (CK) Object Oriented metrics was used in the detection of fault-proneness of the source code of openly available software systems. The authors check the value obtain against the bugs found in the database containing bugs using machine learning and regression algorithms to confirm convenience of the OO metrics for the prediction of fault-prone classes .

In 2017, A. Boucher [9] presented the hybrid self organizing map (SOM) model using source code metrics to find out the fault prone functions present in software module. The authors used Hybrid SOM model for OO software systems to calculate fault prone code at the class level using OO source code metrics which made it easier to prioritize the efforts of the testing team .

In 2014, S. K. Mohapatra [10] proposed a approach for test case reduction using genetic algorithm with different length of chromosome to decrease test suit by finding representative set of test cases that fulfilled the testing criteria.

In 2017, S. Ali, Y. Li, T. Yue, and M. Zhang,[11] proposed multi-objective uncertainty-wise test case minimization approach. The approach focused on to choose a minimum number of test cases for execution by maximizing effectiveness e.g. coverage, limited cost, execution time.

In 2013, A. S. A. Ansari, P. K. K. Devadkar, and P. Gharpure, [12] defines a test suite method which was a effective technique that achieve significant reduction in the test suite and also ensure product quality of the software. It reduced the time and cost of regression testing and also reduces the cost of maintenance activity and effort.

In 2018, O. Banias [13] proposed a dynamic programming algorithm that was apply in software testing domain, generally in the selection of the test cases. The authors defines specific problem present in software testing that is running a subset of test suite from the complete set of test suite and the aim is to maximize the probability of finding potential defects present.

In 2014, K. Choudhary [14] proposed a multi objective optimization that deals with the disagreeing objectives. A multi-objective problem was used to find the solution for all disagreeing objectives. The authors focused on the automatic test data generation. One of the objectives was uniform distribution and another was to maximize the code. The approach covered non-dominance property to maintain sub-population of best fitness value.

In 2016, C. Technology, R. Khan, M. Amjad, and A. K. Srivastava [15] proposed path based testing approach covering all du-paths for a given program. The GA was used for automatic test suite generation and optimization purposed against the accepted a set of inputs and checked for the path coverage.

In 2013, S. Sun, X. Hou, C. Gao, and L. Sun [16] combined test case selection with test case prioritization. The reason of test case selection was to check modified impact of programs and dependencies between the programs. Test cases which were selected during the selection phase were ordered for the prioritization of the test cases.

In 2017,Vandana, Ajmer sigh [17] proposed a multi objective optimization technique for minimizing test cases. The authors finds that the use of meta heuristic algorithms with the optimization can reduce the number of redundant test cases and increase in the accuracy of automated testing.

In 2015, V. Gupta [18] proposed a quantitative research and develop a prediction models which uses bug indicators as models input and performed on open source projects namely Ant and Camel. In the research, the results verified that there was considerable correlation exist between the size metrics or bug indicators metrics such as DIT, WMC, CBO, LOC and bugs. The DIT metric took control in achieving better impact than other bugs predicting metrics such as WMC, CBO and LOC.

In 2018,Rajvir Singh, Anita Singhrova and Rajesh Bhatia [19] proposed an optimized test case generation (TCG) approach. The effective OO metrics were selected and study carried out for the ant-1.7 software. The multivariate linear regression approach was used for generating mathematical model and for giving weights to test paths.

In 2019,Rajvir Singh, Rajesh Bhatia and Anita Singhrova [20] proposed demand based TCG method that selects the test scenarios as per contextual demand in terms of percentage. The optimized test cases were selected to fit within the budget limitations.

This present paper is proposes the minimization of test cases at package level and analyses the applicability of proposed methodology. In the existing techniques, the test suite were generate at the class level.

## III. PROPOSED METHODOLOGY

The diagram of proposed methodology for the prediction of bugs is shown in Figure 1 given below.

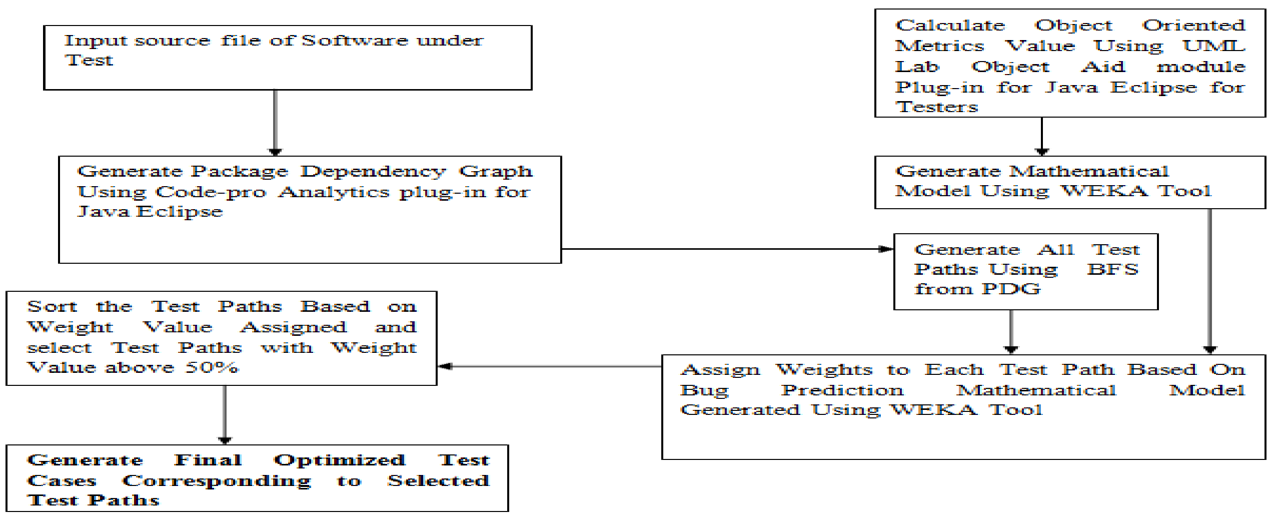


Figure 1 Block diagram of proposed methodology for test case minimization

Select of the effective object-oriented metrics for bug prediction of the classes of *Camel-1.6.1* open source software using WEKA machine learning tool as proposed by Singh et al. [19]. Then after selecting effective metrics the below steps were followed.

The followed steps are:

i. Generate the mathematical model of the selected metrics using WEKA tool.
ii. Input the source file of software under test i.e *Camel 1.6.1*.
iii. Generate package dependency graph (PDG) by selecting the java files using code-pro analytic plug-in for java eclipse.
iv. Calculate the weights of the package by adding the weights of the classes present in the packages that covered by the individual test paths.
v. Generate all paths applying breadth first search (BFS) on PDG.
vi. Assign weights to each of the test path using below proposed equation (1):

$$Weight(TPk) = \sum_{i=1}^{n}(Wp_i) \qquad (1)$$

Where, $Wp_n$ is the weight of $i^{th}$ package covered by the test path and i = 1, 2, 3, …, n. where n is the total number of packages covered by $k^{th}$ test path. Weight ($TP_k$) is the weight of $k^{th}$ test path and k = 1, 2, 3, … m. m is the total number of test paths.

vii. Sort the test paths in decreasing weight values assigned and select the test paths with higher weight values covering 50% of highest weight value.
viii. Generate final test cases corresponds to selected test paths.

The class level CK Object Oriented metrics has been considered for the fault proneness of classes which in turn have been used to calculate weights of packages. The

 

Metrics data was collect from the publicly available and open access repository known as promise repository [4].*Camel1.6.1* have been used for evaluation of proposed methodology.

## IV. IMPLEMENTATION

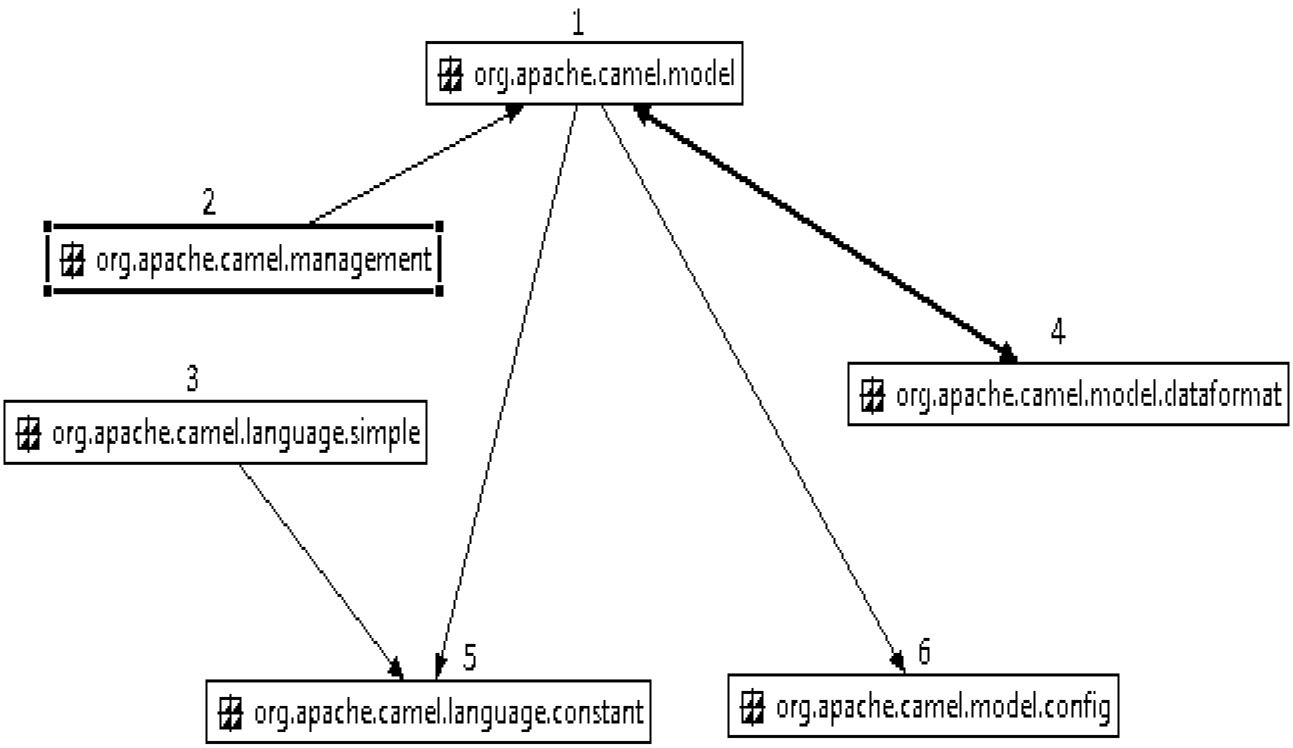


Figure 2 Package Dependency Graph of Camel1.6.1 Module

The implementation of the proposed methodology depicted in Fig1 is as under:

***Step1.*** Select effective Object Oriented metrics finding bugs. The OO metrics namely *wmc, rfc, lcom, ce, npm, loc, bug* have been selected and were used for generating mathematical model using Weka tool. The screen shot of generated mathematical model is shown in Figure. 3.

***Step2.*** Generate PDG of the software module *camel1.6.1* using code-pro analytic in eclipse neon for testers as shown in Figure 2.

From the package dependency graph (PDG) in Figure 2, the test cases were generated using BFS method. The test cases generated are given in Table1below:

Table1. All Test Cases

| Test Case ID | Test Case |
|---|---|
| TC1 | 1,6 |
| TC2 | 1,5 |
| TC3 | 1,4 |
| TC4 | 2,1 |
| TC5 | 2,1,4 |
| TC6 | 2,1,5 |
| TC7 | 2,1,6 |
| TC8 | 3,5 |
| TC9 | 4,5 |
| TC10 | 4,1,6 |

***Step****3.* Select test paths using equation (2) generated by using open source software tool known as WEKA. The selection of the test cases based on the value of weight assigned to each test cases. Weights are assigned to each test cases by using linear regression model "(2)" i.e. the values of the weight that are calculated by using model generated in WEKA for *Camel1.6.1* open source software module.

$$Bug = 0.1758 * wmc - 0.0206 * rfc - 0.0008 * lcom - 0.0932 * npm + 0.0018 * loc - 0.0524 \quad (2)$$

The mathematical model represented in Figure. 3, the value of root relative squared error (R2) is 136.6585%.

The table 2 shows the value of weight assigned to the test cases based on mathematical model generated by WEKA represented by equation "(1)" and "(2)". The tests paths are sorted in decreasing order of the weights are shown in Table

 

3. The sorted test path help in selecting the optimum test paths.

***Step4.*** The package level test cases have been generating resultant to the selection of test path. In this case study of *camel1.6.1*, Table 4 represents the selected test paths which are selected on the basis of weights value assigned to each test path. Test case are selected whose value is greater than 50% of highest weight value because of the limited time and less cost requirements.

Table5 represent the selected test cases and their packages detail and table 6 shows the bug revealed by all selected test cases.

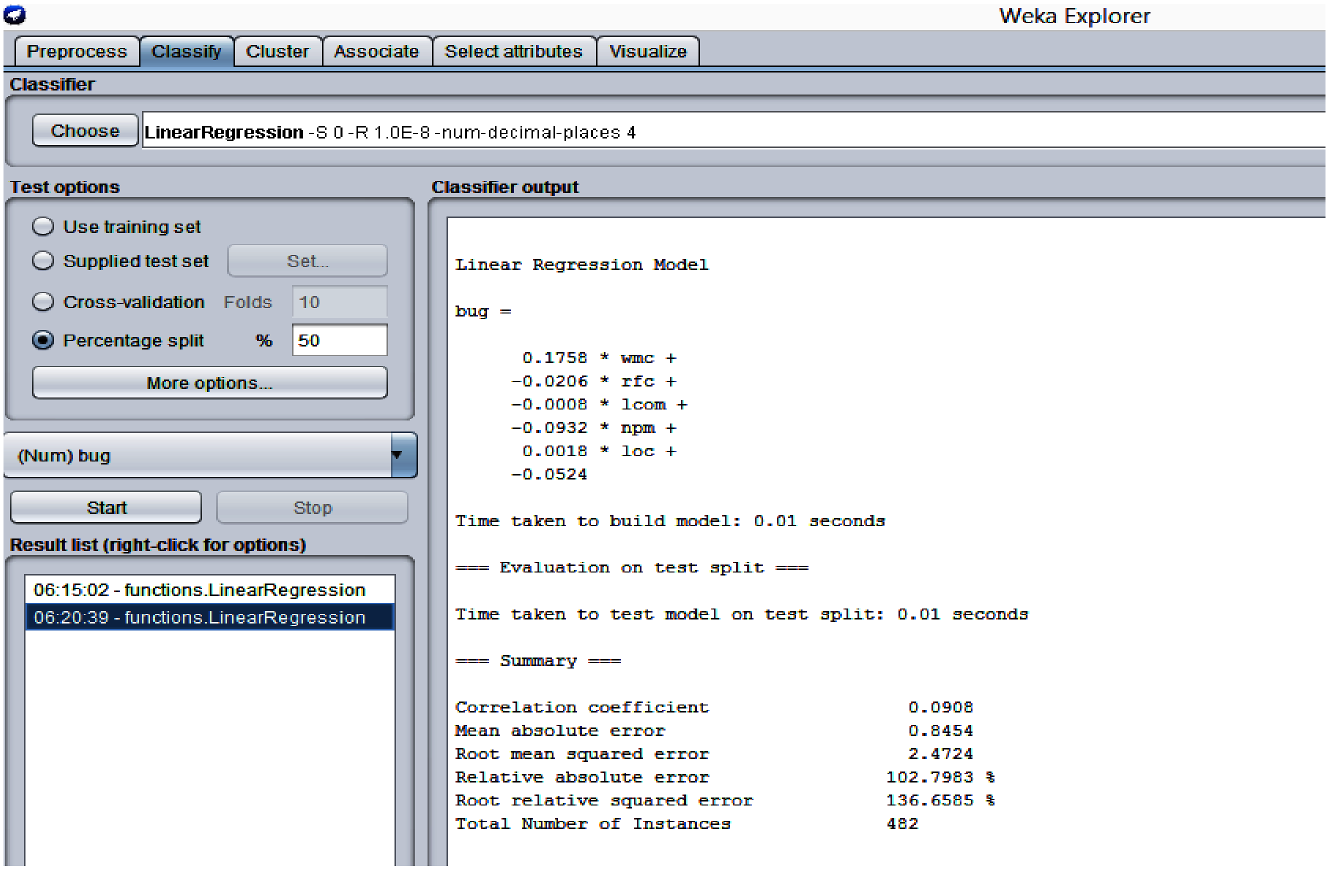


Figure 3 Mathematical model

All the test cases are generated by using the breadth first search(BFS) are shown in these figures given below:

```
path from src 1 to dst 6 are
1 6

--------------------------------
Process exited after 0.07755 seconds with return value (
Press any key to continue . . .
```

```
path from src 1 to dst 5 are
1 5

--------------------------------
Process exited after 0.06059 seconds with return value
Press any key to continue . . .
```

```
path from src 1 to dst 4 are
1 4

--------------------------------
Process exited after 0.03073 seconds with return value 0
Press any key to continue . . .
```

```
path from src 2 to dst 1 are
2 1

--------------------------------
Process exited after 0.07779 seconds with return value 0
Press any key to continue . . .
```

 

```
path from src 2 to dst 4 are
2 1 4

--------------------------------
Process exited after 0.09561 seconds with return value
Press any key to continue . . .

path from src 2 to dst 5 are
2 1 5

--------------------------------
Process exited after 0.03099 seconds with return value
Press any key to continue . . .

path from src 2 to dst 6 are
2 1 6

--------------------------------
Process exited after 0.02176 seconds with return value
Press any key to continue . . .

path from src 3 to dst 5 are
3 5

--------------------------------
Process exited after 0.09629 seconds with return value 0
Press any key to continue . . .

path from src 4 to dst 1 are
4 1

--------------------------------
Process exited after 0.01836 seconds with return value 0
Press any key to continue . . .

path from src 4 to dst 5 are
4 1 5

--------------------------------
Process exited after 0.07608 seconds with return value 0
Press any key to continue . . .

path from src 4 to dst 6 are
4 1 6

--------------------------------
Process exited after 0.03499 seconds with return value 0
Press any key to continue . . .
```

## V. Results and Discussion

The analysis of the results that are obtained by using proposed methodology is discussed below:

- Total faults exposed if all the test cases were executed = 39
- Total number of faults that is covered by the chosen test cases = 31.
- The Fault Exposition Potential (FEP) of the chosen test cases = 80%.
- Execution time reduced by 50%.

Hence, it can be concluded that the proposed methodology is effective methodology in terms of FEP.

Table2. Calculated Weight of the Test Case

| Test Case-ID | Weight Value |
|---|---|
| TC1 | 4.7438 |
| TC2 | 4.4612 |
| TC3 | 10.0074 |
| TC4 | 10.6780 |
| TC5 | 16.3882 |
| TC6 | 10.8420 |
| TC7 | 12.2246 |
| TC8 | 1.4982 |
| TC9 | 5.8742 |
| TC10 | 10.5540 |

Table3. Ordered Test Cases

| Test Case-ID | Weight Value |
|---|---|
| TC5 | 16.3882 |
| TC7 | 12.2246 |
| TC6 | 10.8420 |
| TC4 | 10.6780 |
| TC10 | 10.5540 |
| TC3 | 10.0074 |
| TC9 | 5.8742 |
| TC1 | 4.7438 |
| TC2 | 4.4612 |
| TC8 | 1.4982 |

Table4. Selected Test Cases

| Test Case-ID | Test Cases |
|---|---|
| TC4 | 2,1 |
| TC5 | 2,1,4 |
| TC6 | 2,1,5 |
| TC7 | 2,1,6 |
| TC10 | 4,1,6 |

Table5. Final test cases generated

| Test Case-ID | Test Case | Package Level Test Cases |
|---|---|---|
| T4 | 2,1 | Org.apache.camel.management → Org.apache.camel.model |
| T5 | 2,1,4 | Org.apache.camel.management → Org.apache.camel.model → Org.apache.camel.model.dataformat |
| T6 | 2,1,5 | Org.apache.camel.management → Org.apache.camel.model → Org.apache.camel.language.constant |
| T7 | 2,1,6 | Org.apache.camel.management → Org.apache.camel.model → Org.apache.camel.modify.config |

 

| | | |
|---|---|---|
| T10 | 4,1,6 | Org.apache.camel.model.dataformat → Org.apache.camel.model → Org.apache.camel.modify.config |

Table6. Bug covered by selected Test Cases

| Test Case ID | Test Case | Bug Revealed |
|---|---|---|
| T5 | 2,1,4 | 31 |
| T4 | 2,1 | 23 |
| T6 | 2,1,5 | 23 |
| T7 | 2,1,6 | 23 |
| T10 | 4,1,6 | 12 |

## VI. CONCLUSION

The proposed methodology firstly selected the effective Object Oriented metrics for bug prediction using WEKA tool. The PDG was generated using code-pro for *camel-1.6.1* open source software module. Using BFS test paths were generated. The model is generated using WEKA for the *camel-1.6.1* dataset available at promise repository. Weights values were assigned to each of the generated test case. The final test suite has been generated resultant to selected test paths only which saved time and effort of testing. To fit within the limited time the test cases have been minimized whose FEP = 80% and execution time reduced by 50%. This showed that the proposed methodology is an effective methodology.

## REFERENCES

[1] P. Mandal and A. S. Ami, "Selecting Best Attributes for Software Defect Prediction," no. December 2015, 2019.

[2] S. Puranik, P. Deshpande, and K. Chandrasekaran, "A Novel Machine Learning Approach for Bug Prediction," Procedia Computer Science, vol. 93, pp. 924–930, 2016.

[3] S. Prateek, A. Pasala, and L. M. Aracena, "Evaluating Performance of Network Metrics for Bug Prediction in Software," no. December 2013, 2017.

[4] "Promise repository," 2014. [Online]. Available: http://openscience.us/repo/defect/ck/. [Accessed: 26-Mar-2018].

[5] A. Singh, R. Bhatia, and A. Singhrova, "Taxonomy of machine learning algorithms in software fault Taxonomy of machine learning algorithms in software fault prediction using object oriented metrics prediction using object oriented metrics," Procedia Computer Science, vol. 132, pp. 993–1001, 2018.

[6] M. Akour and L. Abuwardih, "Test Case Minimization using Genetic Algorithm : Pilot Study," 8th International Conference on Computer Science and Information Technology (CSIT), pp. 66–70, 2018.

[7] D. L. A. L. Gupta and K. Saxena, "Software bug prediction using object-oriented metrics," vol. 42, no. 5, pp. 655–669, 2017.

[8] R. Ferenc, "Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction," vol. 31, no. 10, pp. 897–910, 2005.

[9] A. Boucher and M. Badri, "Predicting Fault-Prone Classes in Object-Oriented Software: An Adaptation of an Unsupervised Hybrid SOM Algorithm," IEEE International Conference on Software Quality, Reliability and Security (QRS), pp. 306-317, 2017.

[10] S. K. Mohapatra and S. Prasad, "Minimizing test cases to reduce the cost of regression testing," IEEE international Conference on Computing for Sustainable Global Development (INDIACom), pp. 505-509, 2014.

[11] S. Ali, Y. Li, T. Yue and M. Zhang, "An Empirical Evaluation of Mutation and Crossover Operators for Multi-Objective Uncertainty-Wise Test Minimization," IEEE/ACM 10th International Workshop on Search-Based Software Testing (SBST), Buenos Aires, 2017, pp. 21-27, 2017.

[12] A. S. A. Ansari, K. K. Devadkar and P. Gharpure, "Optimization of test suite-test case in regression test," IEEE International Conference on Computational Intelligence and Computing Research, Enathi, 2013, pp. 1-4, 2013.

[13] O. Banias, "Dynamic programming optimization algorithm applied in test case selection," *International Symposium on Electronics and Telecommunications (ISETC)*, pp. 1–4, 2018.

[14] K. Choudhary and G. N. Purohit, "A Multi-Objective optimization algorithm for uniformly distributed generation of test cases," IEEE International Conference on Computing for Sustainable Global Development (INDIACom), pp. 455-457, 2014.

[15] R. Khan, M. Amjad and A. K. Srivastava, "Optimization of Automatic Generated Test Cases for Path Testing Using Genetic Algorithm," 2016 Second International Conference on Computational Intelligence & Communication Technology (CICT), pp. 32-36, 2016.

[16] S. Sun, X. Hou, C. Gao and L. Sun, "Research on optimization scheme of regression testing," Ninth International Conference on Natural Computation (ICNC), pp. 1628-1632, 2013.

[17] P. A. Vikhar, "Evolutionary algorithms: A critical review and its future prospects," International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC), pp. 261-265, 2017.

[18] V. Gupta, N. Ganeshan and Tarun K. Singhal, "Developing Software Bug Prediction Models Using Various Software Metrics as the Bug Indicators," International Journal of Advanced Computer Science and Applications, Vol. 6, no. 2, 2015.

[19] R. Singh, A. Singhrova, and R. Bhatia, "Optimized Test Case Generation for Object Oriented Systems Using Weka Open Source Software," *International Journal of Open Source Software and Processes*, vol. 9, no. 3, pp. 15–35, Jul. 2018.

[20] R. Singh, R. K. Bhatia, and A. Singhrova, "Demand Based Test Case Generation for Object Oriented Systems," *IET Software*, 2019.

 