Mutation Analysis Technique for Fault Detection in Software Engineering

Adya Sharma, Anita Chandel
M. tech Scholar, Assistant Professor
Engineering college Bikaner

Abstract- Software engineering is a branch of engineering. This branch is related to the development of software product with the help of distinct scientific principles, techniques and processes. The result of software engineering is an effectual and trustworthy software product. A significant software engineering application is known as testing. In testing, the test cases are implemented for detecting faults within the software. The unwanted outcomes can be generated due to the faults within software. A prediction model should be constructed with predictors for the prediction of faulty files. These predictors can be attained either within the project or from other projects. Not only was this anomaly in the design, but the code was actually implemented with the same key alternating between HELP and DELETE functions based on which application was in use. Nowadays, the faults may also occur in the test cases. The test cases are generally utilized to detect faults. In order to detect faults within the software, the mutation algorithm has been implemented in this study. Back propagation algorithm has been implemented for improving the performance of mutation algorithm in terms of fault discovery rate. This algorithm learns from the past experience and determines novel values. Total 10 test cases are used for software testing. The simulation is executed in MATLAB software. The outcomes of simulation depict improved fault detection rate and decreased execution time.

I. INTRODUCTION

Test case prioritization is the approach that is required to arrange a test case in such an order that the effectiveness is increased. The fault detection rate can be measured using this approach. The test cases are ranked and programmed appropriately with the help of test case prioritization. The higher priority test cases are run initially followed by running the lower priority test cases. This result in lessening the time, hard work and cost required in the software testing phase. The speed at which the fault is detected in software through testing approach is an important step. The software can be corrected by removing the identified fault based on the feedback given by the previous step [7].

Regression testing is defined as retesting the software. It is an important application of prioritization technique. This testing method ensures that any new modifications made in the previously tested software does not result in causing new errors. Thus, the modified software is retested here. The cost required to perform regression testing is very high. The test cases are prioritized in the regression testing process to ensure that the vital test cases can be executed initially. The information that is collected from previously executed test cases can prove to be very beneficial for arranging the test cases as per their priority. A clustering method is used to perform test case prioritization in regression testing. The test cases with general assets and parallel capability to detect faults are grouped jointly. For executing the test cases of higher priority, the software testing process that provides minimal cost, effort and time is designed.

Genetic algorithms (GAs) are search techniques. These techniques depend on laws of natural selection and genetics. The decision variables of a search issue are encoded into fixed length alphabets strings of definite cardinality by Genetic algorithms. Chromosomes are the strings. These strings are candidate solutions to the search issue. The alphabets are known as genes. The values of genes are known as alleles. As an instance, a route is represented by a chromosome and a city may be represented by a gene. GAs performs with coding of parameters instead of the parameters themselves unlike conventional optimization methodologies. A measure is required to make difference amid good and bad solutions for developing good solutions and implementing natural selection. It is possible for measure to be an objective function. This is a statistical model or a computer simulation. It can also be a subjective function. In this function, the better solutions are selected by human beings over bad solutions. The virtual fitness of a candidate solution must be determined by the fitness measure in real time. Afterward, genetic algorithm will use this fitness to direct the development of high-quality solutions.

The concept of population is one more significant theory of genetic algorithms. Genetic algorithms are based on the population of candidate solutions in contrast to conventional search techniques. The population size is generally a client-specified parameter. This is a one important factor that affects the scalability and functioning of genetic algorithms. As an instance, small population dimensions may cause early convergence. This results in substandard solutions. However, big population dimensions can cause redundant costs of precious computation time. The development of solutions can

be started for problem searching after encoding the issue in a chromosomal way. A fitness measure is selected to distinguish good solutions from bad solutions. In this process, following steps are applied:

i**. Initialization**: In general, the preliminary population of candidate solutions is generated randomly crosswise the search space. On the other hand, it is easy to include the domain-specified information or other information.

ii. **Evaluation**: The fitness values of the candidate solutions are assessed after population initialization [10].

iii. **Selection**: Additional facsimiles of those solutions are distributed by selection with better fitness values. Therefore, the survival-of-the-fittest method is applied on the candidate solutions. The major thought of selection is to choose higher solutions over the bad solutions. A lot of selection methodologies have been resented to achieve this aim. These methods include roulette-wheel selection, stochastic universal selection, ranking selection and tournament selection etc.

iv. **Recombination**: The components of two or more than two parental solutions are combined by this approach for creating novel, perhaps the improved solutions. This aim can be fulfilled by different methods. The performance is based on an appropriately designed recombination method. The offspring in recombination will not be same as some specified parent. It will rather merge parental features in a new way.

v**. Mutation**: A solution is modified by mutation locally but in random manner unlike the recombination that works on two or more than two parental chromosomes. Also, the mutation is of different types. However, it generally includes one or additional variations. These variations are being made to an individual's feature or features. Hence, a random walk near a candidate solution is performed by the mutation.

vi**. Replacement**: The original parental population is replaced by the offspring population. This population is developed by selection, recombination, and mutation techniques. In genetic algorithms, various replacement methodologies like elitist replacement, generation-wise replacement and steady-state replacement methods are utilized.

vii. Repeat steps 2–6 until a terminating condition is met.

1.1. Mutation Testing

The procedure of adding little faults into programs and determining the efficiency of test case for fault detection is called Mutation testing. In the past few years, several mutation operators were designed. These operators defined the change within the program. Every change generates a novel program. This new program is known as mutant. This program is slightly different from the original one. Mutagenesis is a process that creates a mutant from the real program. The potential to identify mutants is a measure of the efficiency of test case. The mutants which are not identified by the test case are known as living mutants.

The proportion of killed mutants to the total amount of mutants is called mutation score. This is a measure of mutation's effectiveness. Choosy mutation can considerably decrease the outlay of mutation scrutiny.

II. LITERATURE SUVERY

Xiaoxing Yang et.al (2015) explained previous work of the construction and also whether the measure of performance model was beneficial software defect prediction model construction an idea of direct optimization. For software defect prediction one is a new learning-to-rank approach to practical data sets and for predicting the order software modules based on the predicted number of defects as compared to the learning-to-rank technique alongside algorithms are the two aspects included in this work. For the ranking task the experimental studies demonstrates the efficiency of straightly improving the functioning of the model measuring for the learning-to-rank technique for building fault prediction systems [11]

Shaik Nafeez Umar et.al (2013) explained a new statistical model. In this work, the efficiency of proposed model for predicting the faults of future software products was discussed. This study utilized 20 earlier launched data points of software project and five parameters. These data points and parameters were used to construct the proposed model. Descriptive statistics, correlation and multiple linear regression models with 95% confidence intervals (CI) were applied to construct this model [12]. The R-square value of 0.91 and its Standard Error of 5.90% were obtained in the constructed statistical model. The Software testing fault prediction model was utilized for fault prediction at a variety of testing projects and functional deliveries. The accuracy between original and forecasted faults was 90.76% as per the analysis.

Muhammad Dhiauddin et.al (2012) explained the preliminary endeavor to construct a defect prediction model. This model was constructed to detect faults in system testing. A self-determining testing team performed the task of system testing. The main motive behind the construction of defect prediction model was to provide an early quality sign of the software incoming system testing [13]. This model was also constructed to help the testing team for managing and controlling the behavior of test implementation. The possible predictors in model construction were determined by identifying and analyzing the metrics gathered from earlier stages of system testing. Afterward, in order to produce some numerical equations, the chosen metrics were applied in regression analysis. Numerical equation having p-value below 0.05 with Rsquared and above 90% with R-

squared (adjusted) was chosen as the preferred forecasting model for system analyzing faults. Novel projects were utilized to validate proposed model. The obtain results confirmed that the proposed model was appropriate for real time execution.

Mrinal Singh Rawat, Sanjay Kumar Dubey (2012) proposed the underlying cause of failure by the software faults. The faults within software could degrade the quality of software product. In this scenario of cut-throat rivalry, it was required to attempt continuously for controlling and minimizing the faults in software engineering. Money, time and resources are used in these efforts. Improve software quality and productivity contributing aspects suggesting measures are identified by this paper. This work also demonstrated the way to implement the several defect prediction models to reduce the size of faults. [14].

Christopher Henard, (2013) explained large number of customization and big economics pushed developers to design software product line in this paper. On addition of more features it reuses its assets. The feature model and allow tailored software products represents many constraints. Thousands and billions of software products are contained in it. Because of large size of products, product line is challenging as a result. Limited product suites are selected based on the feature model existed by this technique. These types of faults are detected using test cases. Specifically, two mutation operators were presented in this work. The main aim of these operators was to obtain mutants from a real feature model. These operators also evaluated the potential of the produced real test case for killing the erroneous feature model. Hence, validation of the significance of same-driven product line test are based on the experimental results demonstration that is not same test cases with improved mutant discovery potential as compared to identical ones [15].

Jan Peleska, (2013) explained one of the leading technologies is based on the model testing. In this work, the important aspects for efficient manufacturing application of MBT were explained. Both technical and a decision-making prospective were considered for this purpose. The methodologies for automatic test suites, test data and test process production for real-time reactive concurrent systems were also explained in this work with former view. In MBT for testing teams their knowledge initiated MBT schemes. For an improvement of the acceptance and effectiveness of MBT many specific problems arise [16].

III. RESEARCH METHODOLOGY

Complex and faulty test information is generated using mutation analysis model such that automated testing can be performed. In the presence of faulty data, performing efficient testing and analyzing the robustness of system becomes difficult. In this research, the Boltzmann learning algorithm is used to detect the faults existing in the generated test cases. Once the faults are removed, the testing efficiency of the system is increased.

The visible units are held to the values given by the teacher in the initial layer. The pair-wise potentials are used in the second layer to forward the underlying components to the hidden layer. The stochastic recurrent neural networks are the two layers in which binary neurons are arranged. Every neuron present in the visible layer is linked to all the hidden neurons so that the neurons can be visible. There is a link between each neuron present in the hidden layer. A binary column vector is denoted by 'v' and it contains the states of visible neurons. It is possible to denote hj similar to the vector 'h' of a hidden state. The systems of symmetrically connected units are known as Boltzmann machines which can settle the on or off conditions of stochastic decisions. For discovering the complex distributions, a simple learning algorithm is applied on the observed information. It is imperative to perform learning in Boltzmann machines for performing different scientific tasks. The weights on connections and thresholds to represent are settled to represent the cost of functions. Also, the inference related issues can be handled through this process. Inference is a tool that is commonly used to improve the issues in which the combinatorial issues arising in NP finish or hard issue classes of Boltzmann machines can be handled. The expectations of one unit as well as correlations are needed in between the two units that are performing learning in Boltzmann machines.

Fault prediction is the technique that is used to predict the percentage of faults in test cubes. The faults can be detected from text cases using the learn-to-rank algorithm. This algorithm is based on three important steps. In the initial step, the population is selected. The next step is to calculate the mutation value. In the final step, the fitness value is calculated. The fitness value is calculated based on the randomly chosen final population value. The system performs learning with the help of values and derivations of new values back propagation technique. The selection of population value is not random. Based on few system conditions, the back propagation algorithm is applied here.

Steps of Proposed Algorithm

Following are the various steps of the proposed algorithm for software fault detection :-

1. The software is taken as input from which faults needs to be detection. In the software various software modules are taken as input like signup, login etc.

2. Input the test case of each module for the fault detection from the input software.

3. The test case number and test case range will be the input to the Boltzmann learning for the software defect prediction.

4. Apply formula Error =Desired- Actual errors for calculating error in each module.

5. The step 4 will be repeated until all the defects from the software will be detected.

6. Display defects predicted in every iteration from the software.

IV. RESULT AND DISCUSSION

Matrix laboratory is MATLAB: tool. Providing numerical computation and visualization data is an interactive program. It is a very powerful tool. This tool is extremely helpful for all disciplines of science and engineering with the help of its programming capabilities. With the feature of graphical user interface MATLAB helps developing applications.

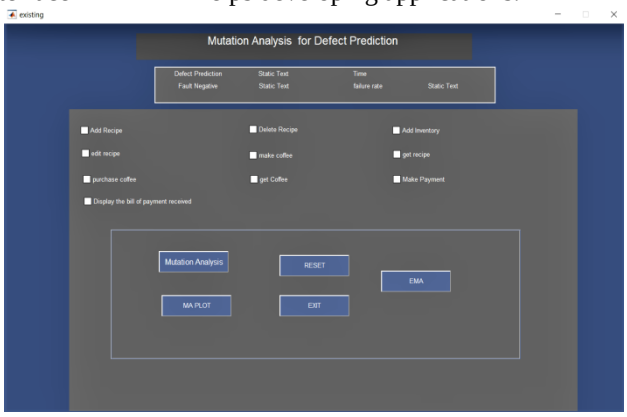

Fig.1: Mutation testing interface

As shown in figure 1, the mutation testing is the type of testing. In this testing, test cases are generated which test the software. The test cases which are generated have the faults which reduce efficiency of fault prediction

Fig.2: Selection of test cases

Figure 2 shows the implementation of proposed algorithm. Boltzmann learning algorithm is used for mutation analysis. The Boltzmann learning algorithm will learn from the past knowledge and determine novel values

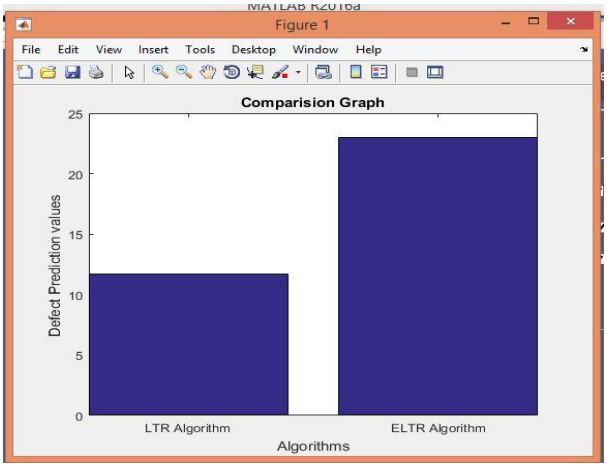

Fig.3: Defect Prediction Analysis

Figure 3 shows the comparison of Mutation and E-Mutation values to analyze their performances. The E-Mutation approach has high defect prediction value as compared Mutation algorithm

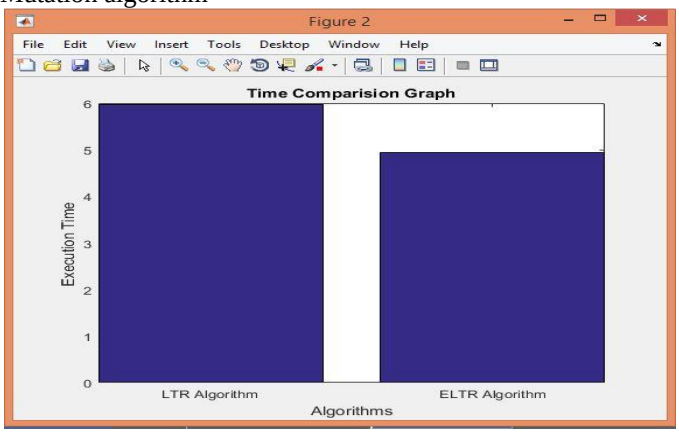

Fig.4: Time Analysis

Figure 4 shows the comparison of Mutation and E-Mutation values to analyze their performances. The E-Mutation algorithm has low execution value than Mutation approach

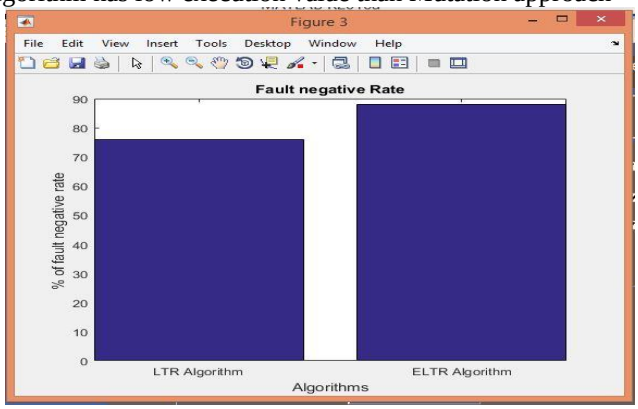

Fig.5: Fault Negative Rate Analysis

Figure 5 shows the comparison of Mutation and E-Mutation values to analyze their performances. The E-Mutation algorithm has high fault negative rate as compared Mutation algorithm

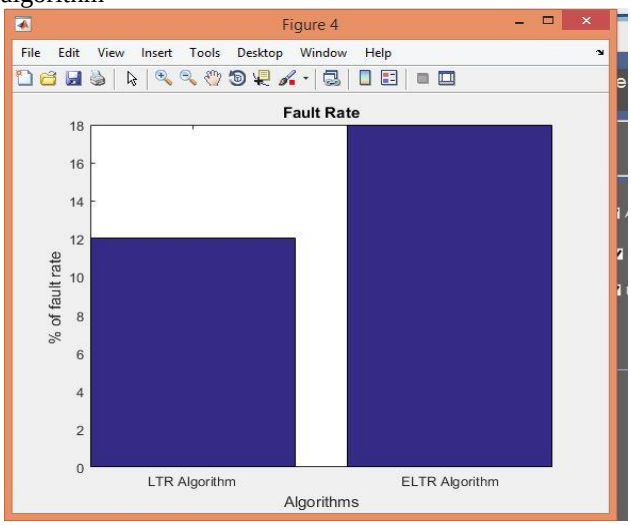

Fig.6: Fault Rate Analysis

Figure 6 shows the comparison of Mutation and E-Mutation values to analyze their performances. The E-Mutation algorithm shows high rate than Mutation algorithm

V. CONCLUSION

A testing method called fault detection is implemented for detecting faults within the software or within the input test cases. For the faults in the software the mutation is known as the algorithm. Reduction in the fault detection rate is the algorithm that chose the population randomly. From the previous experience and driving new values, back propagation algorithm is implemented from where the system learns in this work. For testing the procedure a test case is a set of procedure. The software delivery also gets affected by this. Improvement in the defect detection rate and reduction in the execution time are lead by this. O the basis of the bio-inspired techniques for the fault detection rate, technique will be proposed in future. This algorithm learns from the past experience and determines novel values. Total 10 test cases are used for software testing. The simulation is implemented in MATLAB software. The outcomes of simulation depict improved fault detection rate and decreased execution time.

VI. REFERENCES

[1]. Gaurav, Kestina Rai "Software Testing Techniques for Test Case Generation" International journal of Advanced Research in Computer Science and Software Engineering 2013 pp 261-265.

[2]. John E. Bentley, Wachovia Bank, Charlotte NC, "Software Testing Fundamentals—Concepts, Roles, and Terminology", 2009

[3]. Chandana Bharati1, Shradha Verma, "Analysis of Different Regression Testing Approaches", International Journal of Advanced Research in Computer and Communication Engineering Vol. 2, Issue 5, May 2013

[4]. Gaurav, Kestina Rai "Software Testing Techniques for Test Case Generation" International journal of Advanced Research in Computer Science and Software Engineering 2013 pp 261-265.

[5]. K. Pohl, G. B¨ockle, and F. J. van der Linden, "Software Product Line Engineering: Foundations, Principles and Techniques", Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.

[6]. P. Clements and L. Northrop, "Software Product Lines: Practices and Patterns", Addison Wesley, Reading, MA, USA, 2001.

[7]. M. Mendonca, A. Wasowski, and K. Czarnecki, "Sat-based analysis of feature models is easy" in Proceedings of the 13th International Software Product Line Conference, ser. SPLC '09. Pittsburgh, PA, USA: Carnegie Mellon University, 2009, pp. 231–240.

[8]. G. Perrouin, S. Sen, J. Klein, B. Baudry, and Y. L. Traon, "Automated and scalable t wise test case generation strategies for software product line" in ICST. IEEE Computer Society, 2010, pp. 459–468.

[9]. Mangal, B. S, "Analyzing Test Case Selection using Proposed Hybrid Technique based on BCO and Genetic Algorithm and a Comparison with ACO", IJCA, 2012.

[10]. Suman and Seema, "A Genetic Algorithm for Regression Test Sequence Optimization", International Journal of Advanced Research in Computer and Communication Engineering Vol. 1, Issue 7, September 2012

[11]. Xiaoxing Yang, Ke Tang, Senior Member, IEEE, and Xin Yao, "A Learning-to-Rank Approach to Software Defect Prediction", IEEE TRANSACTIONS ON RELIABILITY, VOL. 64, NO. 1, MARCH 2015

[12]. Shaik Nafeez Umar, "Software Testing Defect Prediction Model- A Practical Approach", IJRET: International Journal of Research in Engineering and Technology, Volume: 02 Issue: 05 | May-2013

[13]. Muhammad Dhiauddin, Mohamed Suffian, Suhaimi Ibrahim, "A Prediction Model for System Testing Defects using Regression Analysis", International Journal of Soft Computing And Software Engineering (JSCSE) e-ISSN: 2251-7545 Vol.2,o.7, 2012

[14]. Mrinal Singh Rawat, Sanjay Kumar Dubey , "Software Defect Prediction Models for Quality Improvement: A Literature Study", IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 5, No 2, September 2012 ISSN (Online): 1694-0814

[15]. Christopher Henard, Mike Papadakis∗, Gilles Perrouin, Jacques Klein, and Yves Le Traon "Assessing Software Product Line Testing via Model-based Mutation: An Application to Similarity Testing", 2013

[16]. Jan Peleska, "Industrial-Strength Model-Based Testing - State of the Art and Current Challenges", 2013