International Journal of Computer Science & Communication (ISSN: 0973-7391)
Volume 10 • Issue 1 pp. 250-257 Sept 2018 - March 2019 www.csjournals.com

A Faster Software Fault Prediction using White-Box Testing (LT) and Black-Box Testing (BVA) Techniques

Seema Rani [1], Deepali Gupta[2]
Research Scholar, CSE, M.M.University, Sadopur , Ambala
Associate professor, CSE, M.M.University, Sadopur , Ambala.

.

ABSTRACT

Software development plays a crucial role in today's world. The goal of software development is to create high-quality software. High-quality software possesses attributes like being fault-free, cost-effective and reliable for end users. Testing the software after development plays a significant role in developing defect-free and quality software but consumes almost half the time and resources allocated to develop it. To reduce development time and cost, efficient allocation of resources is necessary, and software defect prediction models can help by giving prior knowledge about defective modules. Throughout the testing process, test cases are executed to detect faults in the software. The purpose of testing software is to run the program to uncover errors. In the software development life cycle, testing is a vital activity for finding all defects. Software testing is essential for minimizing faults in existing software products. Thorough and successful testing reduces overall system costs. Software companies hire testing and quality assurance staff to perform testing activities. One of the main topics of investigation is software defect prediction. A key component of software engineering is software fault prediction. It lowers expenses overall, time, and effort makes a major contribution to the startup and profitability of the company by guaranteeing client happiness. Over the years, this area has drawn a lot of academics in an effort to raise the general standard of the program. To predict software faults, this research proposes testing using white box and black box testing techniques. Among different testing methods, loop testing from white box testing and Boundary Value Analysis from black box testing are selected in this research for Software fault prediction.

***Keywords*:** Software Fault prediction techniques, Software Testing, Software quality, Loop Testing, Boundary Value Analysis, Software defects.

I INTRODUCTION

The practice of identifying potential defect-prone areas in a software system is known as software fault prediction [1]. By using software fault prediction models early in the software lifecycle, practitioners can concentrate their testing resources so that components that have been recognized as "prone to defects" receive more thorough testing than other components of the software system [2]. This result in lower labor expenses during development and less work required for maintenance [3].

Software fault prediction is the most important method which has been researched extensively over the years. Software fault prediction helps in cost efficiency and time efficiency that adds to the quality assurance the quality of software [4]. Faults can occur due to any design issue, functionality error or code error. A crucial step in software engineering is software defect prediction, which helps to increase software quality and assurance while saving money and time. Software defect prediction models indicate if a defect is there or not. Because software defect prediction allows for distinct software quality and monitoring assurances to be increased, many researchers have been motivated to provide alternative models with a project or cross-project. There are two ways to build

a software defect prediction model: supervised learning and unsupervised learning. The problem with supervised learning is that the software defect prediction model cannot be trained without prior data or known results. Even when the model's training within the project is done correctly, other new ventures will find it challenging to comprehend. To solve the difficult challenge while training on a new project, researchers can use a variety of publicly available datasets, such as PROMISE, Eclips, and Apache, for free. A key phrase for software dependability is software testing. Testing gives the program the true structure and validity it needs to function well in operational settings. By using various software testing techniques, error detection and error correction need to be performed. Various methods need to be found to decrease the time and effort of software testers. Software testing's primary objective is to systematically test the program under controlled circumstances in order to confirm its quality. A further objective is to ensure that the program is accurate and comprehensive, and lastly, it discloses faults that have not yet been found [5]. In software engineering, software testing is a broad area, which includes specification, design and implementation, maintenance, process and management issues as well. The goal is to evaluate research on comparing the efficacy of defect prediction approaches in software testing. The Research paper is organized as: Section I covers introduction part. Section II describes the software fault prediction process. Section III explains software testing metrics. Section 1V explains the role of testing in software fault prediction. Section V outlines the methodology and results. Section VI concludes the paper.

II SOFTWARE FAULT PREDICTION PROCESS

A Fault refers to a problem or imperfection that causes software to be unable to execute its intended job properly. Faults lead to failures in software systems. Fixing faults in later stages of the software development life cycle requires additional time and rework. This rework also incurs extra costs. Therefore, identifying faults early in development is crucial to reduce cost [6]. By more effectively concentrating quality assurance efforts on particular modules, early problem identification can help to enhance the software quality.

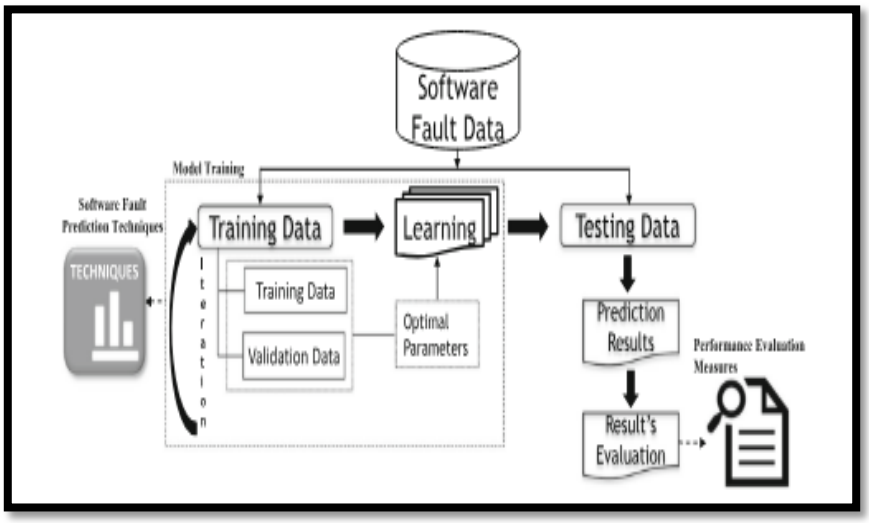

Figure 1: Software Fault Prediction Process(SFP)

An overview of the software fault prediction mechanism is provided in Figure 1. Above figure illustrates the three key elements of the SFP process: performance evaluation metrics, faulty

datasets, and SFP methodologies. First, information about software faults is gathered from software project repositories that hold information about the project's development cycle, including source code. Next, data regarding errors is obtained from the related error repositories. Then information about faults is gathered from the associated fault repositories. Next, the values of several software metrics—like cyclomatic complexity and LOC—are obtained; they act as independent variables. The dependent variable is the necessary fault information for the fault prediction, such as quantity of faults (both faulty and non-faulty). Various Statistical methods are typically employed in the construction of fault prediction models. Pre-processing is used to scale the data, eliminate noise, and extract features [7]. Applying to every kind of defect prediction model is not required [8]. New cases are produced to train the defect prediction model once the cases have undergone pre-processing. Finally, a variety of performance evaluation metrics, including accuracy, precision, and recall are used.

III SOFTWARE TESTING METRICS

To have an efficient and successful software quality assurance process, developers often need to assess the quality of the software artifacts that are currently under development. For this reason, software metrics have been introduced. Metrics allow for the quantitative analysis and quality assessment of software projects. A metric uses numbers to indicate how much of a certain attribute a system, system component, or process possesses. Software metrics are measurements of certain features or attributes of software. Software metrics are frequently employed to assess a program's capacity to accomplish a predetermined objective. Software testing metrics are measurable measures of the process's advancement, effectiveness, productivity, and general state [9]. Metrics for software testing are used to make the process of software testing more effective and efficient. Each software metric is associated with specific functional aspects of the software project, such as coupling, cohesion, inheritance, code modification, etc., and is generally used to reflect an external quality trait, such as fault-proneness, reliability, or testability. Three categories comprise software testing metrics:

1. Process Metrics: Process metrics describe the features and functioning of a project. The Software Development Life Cycle (SDLC) process cannot be improved or maintained without these components. Supervisors can gain important insights into the effectiveness of the process by closely examining these indicators, which will help them make decisions that will improve the process's overall performance. This in turn makes it easier to create new policies and tactics intended to maximize the various stages of the process.

2. Product Metrics: Product metrics describe a product's dimensions, functionality, quality, and complexity. By leveraging these features, developers can raise the caliber of their software development. By using these measures, businesses can assess factors like customer satisfaction, the frequency of faults, the quantities of their products, and other relevant elements. This evaluation supports improvements and raises the product's overall quality by helping to understand its benefits and shortcomings.

3. Project Metrics: Project metrics are employed to evaluate the overall caliber of a project. It is used to calculate expenses, productivity, and fault in a project, as well as to estimate resources and deliverables. Project managers may keep an eye on the project's development, spot any obstacles,

and put the right plans in place to make sure it succeeds by analyzing these indications. These metrics enable better project outcomes and well-informed decision-making by offering useful data on the effectiveness and competence of resource allocation, time management, and cost control.

IV TESTING IN SOFTWARE FAULT PREDICTION

Software fault prediction has been an important part of Software Engineering for over 30 years. It is frequently used to find defective software modules using software measurement data and is an essential part of software quality assurance. In today's context, software development is highly versatile and generates enormous datasets, so detecting faults early on is essential. It's estimated that identifying one third of software faults early can significantly reduce rework. It's necessary to find similarities between faults in newly developed software and existing faults. There are many software development methods that follow a general flow - starting with problem definition, requirements gathering, system design and finally implementation. Software testing is carried out to confirm and validate the created product after development. Automated or manual testing is possible [10]. Since manual testing is a time-consuming and slow process, it is done statically. Early in the life cycle is when this testing is conducted. Walkthrough, Informal Review, Technical Review, and Inspection are some examples of several manual testing techniques [11]. Testing is completed in automated testing by having the tester run the script on the testing tool. Automated testing is also known as dynamic testing. Correctness testing, performance testing, reliability testing, and security testing are other subcategories of automated testing. White box testing and black box testing are two additional categories for correctness testing.

- **White box testing:** White box testing is an extremely efficient method of finding and fixing issues since it allows for the early detection of bugs, mistakes, defects, or faults before they become problematic. This approach can be succinctly described as software testing with program coding and internal structure understanding. A tester must be completely conversant in source code. White box testing's drawback is that it wastes resources because it requires a lot of resources to do for major program types [12]. Condition testing, data flow testing, basic path testing, and loop testing are further subcategories of white box testing.
- **Black Box testing:** Testing software using output requirements alone, without any understanding of the program's internal structure or coding, is known as "Black Box" testing. Here, the users are unaware of internal functioning of the software used to produce the results. White box testing is further categorising into equivalence class partitioning, boundary value analysis, cause effect, comparison testing, model-testing.

V METHODOLOGY AND RESULTS

In this research, we used loop testing technique from white box testing and boundary value analysis testing technique from black box testing on a selected data to compare the effectiveness of both testing techniques in terms of fault prediction. A white box testing method called loop testing concentrates on the reliability and validity of loop constructions. A white box testing method called loop testing concentrates on the reliability of loop constructions. Software or programs with loops are easy to test as long as there are no dependencies between the loops or between the loop and the code it contains. Loop Testing can fix the loop repetition issues, reveal performance and identify loops initialization problems. From black box testing, boundary value analysis (BVA) tests the application at both the higher and lower value limits that are anticipated. Boundary value analysis focus on the boundaries of equivalence classes, as boundary values often reveals faults. In BVA testing test cases choose boundary values that are close to them like just below them or just above

them. Boundary conditions are crucial for testing since defects can be introduced at the boundaries relatively quickly. We compare the performance of software prediction. The experiments were performed using MATLAB-R2015 software. We had compared the prediction of faulty n non faulty files over 350 different data files from kaggle. Data scientists and machine learning professionals can connect online with kaggle. Users can look for and exchange data sets, conduct research, and build models in an online data science environment with kaggle. This dataset is referred to as a public dataset because it is readily accessible. Each piece of data includes information about faults and is made up of several program instances and modules. Every dataset included biased information with both faulty and non-faulty examples. For the purpose of predicting software defects, the data is chosen at random. The main objective of research is to find faulty and non-faulty data by using the Loop Testing and BVA techniques. To analyze the effectiveness of white box and black box testing techniques in terms of fault prediction, we had compared the both techniques loop testing and boundary value analysis.

A matrix with the counts of cases sorted by the actual and expected class is called a confusion matrix. It is a 2 × 2 matrix for the classification problem (as shown in Table 1). The performance of the classifier can be assessed using the confusion matrix and the derived metrics; typical indicators are as follows:

Table 1: Confusion Matrix

Actual/Predicted	**Negative**	**Positive**
Negative	True negative (*TN)*	False positive(*FP*)
Positive	False negative (*FN)*	True positive(*TP*)

Recall is the percentage of real positive class instances that are appropriately classified as belonging to the positive class. The fraction of accurately predicted fault-free modules is measured by recall.

$$\text{Recall} = \frac{TP}{TP+FN}$$

Precision is the percentage of positive class instances that are in fact in the positive class as expected. The precision metric computes the percentage of accurately identified faulty modules among the fault-prone modules.

$$\text{Precision} = \frac{TP}{TP+FP}$$

The fraction of real negative class instances that are mistakenly allocated to the positive class be represented by the term false positive rate (FPR). The percentage of fault-free modules that are projected to be faulty is known as the false positive rate.

$$\text{FPR} = \frac{FP}{FP+TN}$$

The false negative rate is the ratio of a faulty module that is projected to be non-faulty to the total number of modules that are predicted to be faulty.

.

$$\text{FNR} = \frac{FN}{TP+FN}$$

International Journal of Computer Science & Communication (ISSN: 0973-7391)
Volume 10 • Issue 1 pp. 250-257 Sept 2018 - March 2019 www.csjournals.com

Accuracy is the percentage of instances that are correctly classified. The likelihood that fault-prone modules will be appropriately categorized is measured by accuracy. However, it offers no information regarding modules that are mistakenly categorized as fault-free.

$$\text{Accuracy}= \frac{TN+TP}{TN+TP+FN+FP}$$

For analysis different files were uploaded. All thesis files are of different sizes. We had used feature vectors like loc, if count, class count, clause count etc.

1. $For\ i = 1\ to\ total_{feature}.Count$ *// Total number of features in the file*
2. $Lower_{LOOPVALUE} = loop - (1 - 30)\%$ *// Lower bound of loop*
3. $Upper_{LOOPVASLUE} = loop + (1 - 30)\%$ *// Upper boundary of loop*
4. $If\ feature_{value_i} > Lower_{LOOPVALUE} \&\& feature_{value_i} < Upper_{LOOPVALUE}$ *// Test condition*
5. $Non_{faulty} + +$ *// Non faulty if satisfied*
6. $Else$
7. $Faulty + +$ *// Faulty if not satisfied*

$$where\ Loop\ Testing = \frac{\sum_{i=1}^{n} Boundary_i}{n} \quad (1)$$

8. $for\ i = 1\ to\ total_{feature}.Count$
9. $Lower_{Boundary} = Boundary - (1 - 30)\%$
10. $Upper_{Boundary} = Boundary + (1 - 30)\%$
11. $If\ feature_{value_i} > Lower_{Boundary} \&\& feature_{value_i} < Upper_{Boundary}$
12. $Non_{faulty} + +$
13. $Else$
14. $Faulty + +$

$$where\ Boundary\ Value\ Analysis = \frac{\sum_{i=1}^{n} Boundary_{value_i}}{n} \quad (2)$$

A total of 100 test cases has been set and evaluated for the processing of both loop and boundary value testing. A total of 350 files were tested for every test case and hence a total $350 \times 100 = 3500$ units of tests were implemented to identify whether the file is faulty or not. The evaluation has been made in such a manner that out of 100 cases, if the majority says that the file passes the test case, then the file is assumed to be non-faulty else the file is set to be faulty. This method is adopted because there is no prior ground truth available for the processing. If there is any ground truth, then the analysis has not to be radical as there is something already present for the processing. This part is done in order to process the data for the training of the machine learning mechanism. Following figure shows the result structure of all training mechanism.

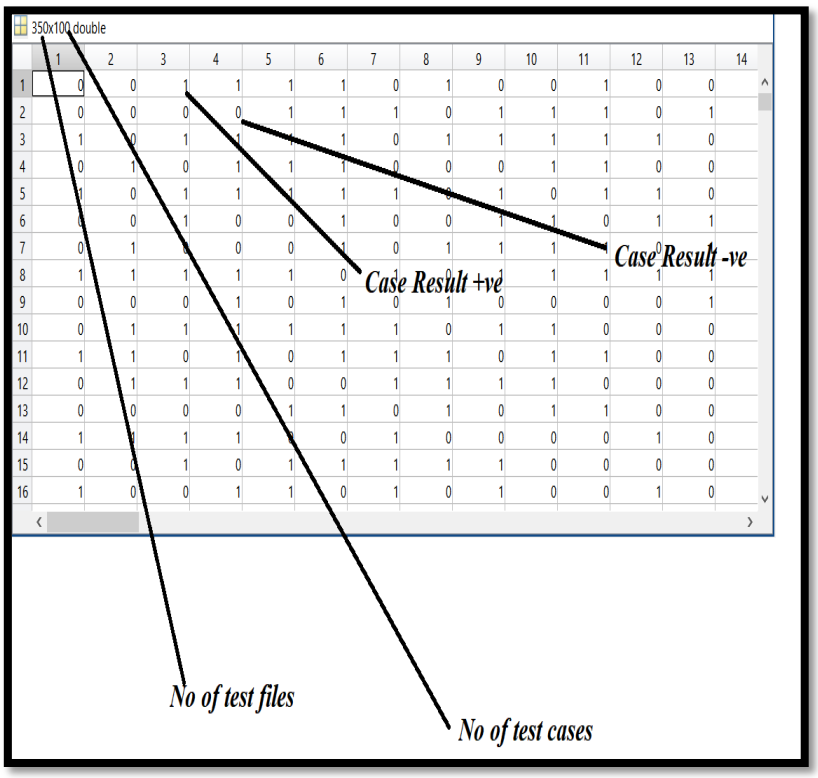

Figure 5: The result structure of all training mechanism.

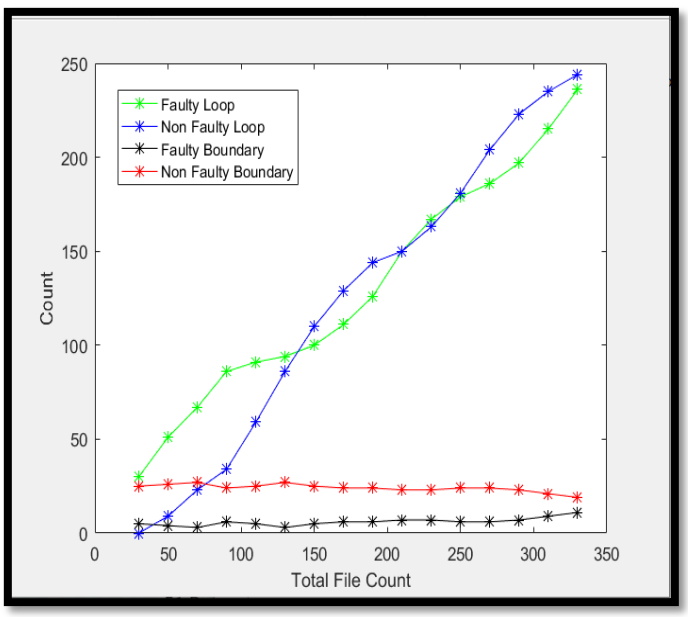

Figure 6: Comparison of Count of faulty files and non-faulty files in terms of fault prediction predicted by Loop testing and BVA.

In above figure green and blue lines shows the faulty files and non-faulty files predicted by loop testing. A Black and Red line shows the faulty files and non-faulty files predicted by Boundary value analysis (BVA).

From the above calculations, it has been observed that loop testing technique for software faults

prediction in terms of count of faulty and non-faulty files is higher than boundary value analysis testing techniques. We also analyzed that fault prediction and fixing fault in the requirement stage is less as compared after delivery stage. So finding the fault in early stages is necessary to reduce the cost.

7. CONCLUSIONS

White-box testing and black-box testing are the two main testing approaches are used in this research. Software testing is carried out throughout the programming improvement cycle and is also done to construct quality programming. It is critical approach for the refine and evaluation of product framework standard. It is very impractical to discover all the mistakes in the program. Software Testing can never be tastefully finished in light of the information area from client. Testing recognizes the mistakes in framework however don't exhibit framework is blunder free. Testing is most basic piece of the Software development Life cycle. Testing is a process to evaluate the quality of software. To test any software different types of testing are used. There is scope for automation in the activities of testing but tester's experience is important for successful testing. In this research, we have compared the both testing techniques Loop testing (white box testing) and boundary value analysis (black box technique) for software fault prediction. By applying both Software testing techniques i.e. Loop Testing and boundary value analysis testing on different data files, it has been observed that loop testing technique for software faults prediction in terms of count of faulty and non-faulty files is higher and better result than boundary value analysis testing techniques. The obtained result supports our claim of the importance of comparison.

REFERENCES

[1] Kitchenham, B.A. Guidelines for Performing Systematic Literature Review in Software Engineering; Technical Report EBSE-2007-001; Keele University and Durham University: Staffordshire, UK, 2007.
[2] Catal, C.; Diri, B. A Systematic Review of Software Fault Prediction studies. Expert Syst. Appl. 2009, 36, 7346–7354.
[3] Radjenovic, D.; Hericko, M.; Torkar, R.; Zivkovic, A. Software fault prediction metrics: A Systematic literature review. Inf. Softw. Technol. 2013, 55, 1397–1418.
[4] He, H.; Garcia, E.A. Learning from Imbalanced Data. IEEE Transactions on Knowledge and Data Engineering. IEEE Trans. Knowl. Data Eng. 2009, 21.
[5] Mohd. Ehmer Khan, "Different Forms of Software Testing Techniques for Finding Errors," IJCSI, Vol. 7, Issue 3, No 1, pp 11-16, May 2010.
[6] A. Bacchelli, M. D'Ambros, and M. Lanza. Are popular classes more defect prone? In Proceedings of the 13th International Conference on Fundamental Approaches to Software Engineering, FASE'10, pages 59– 73, Berlin, Heidelberg, 2010. Springer-Verlag.
[7] S. Kim, H. Zhang, R. Wu, and L. Gong. Dealing with noise in defect prediction. In Proceeding of the 33rd international conference on Software engineering, ICSE '11, pages 481–490, New York, NY, USA, 2011. ACM.
[8] T. Jiang, L. Tan, and S. Kim. Personalized defect prediction. In Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on, pages 279–289, Nov 2013.
[9] A. Campan, G. Serban, T. M. Truta, and A. Marcus, "An algorithm for the discovery of arbitrary length ordinal association rules," DMIN, vol. 6, pp. 107-113, 2006.
[10] A. Sethi, "A review paper on levels, types& techniques in software testing", International Journal of Advanced Research in Computer Science, vol. 8, no. 7, pp. 269-271, 2017.
[11] S.M.K Quadri and Sheikh Umar Farooq,"Software Testing-Goals,Principles and Limitations," International Journal of Computer Applications, Volume 6-No.9,September 2010.
[12] Srinivas N and Jagruthi D 2012 Blackbox and Whitebox Testing Techniques - A Literature Review International Journal of Embedded Systems and Applications (IJESA) 2(2) 29-50.