History Research Journal

ISSN: 0976-5425
Vol-5-Issue-4-July-August-2019

A Study on Bug Prediction in Determining The Software Quality

D. Siri
Research Scholar
Shri JJT University
Rajasthan
Assistant professor
Dept of IT, MRECW
dharmapuri.siri@gmail.com

DR. PRASADU PEDDIA

Assistant professor

Shri JJT University

Rajasthan

DR. D. MURALI

PROF & HOD

VEMU INSTITUTE OF TECHNOLOGY, P

KOTHAKOTA,CHITTOOR AP

ABSTRACT:

Software quality is a field of study and practice that describes the desirable attributes of software products. The performance must be perfect without any defects. Quality of the developed software depends on its bug free operation. Although bugs can be introduced in any phase of the software development life-cycle but their identification in earlier phase can lead to reduce the allocation cost of testing and maintenance resources. Software defect prediction studies advocates the use of defect prediction models for identification of bugs prior to the release of the software. Use of bug prediction models can help to reduce the cost and efforts required to develop software. There have been many research efforts to improve accuracy in software defect prediction using a variety of classifiers and data pre processing techniques. Defect prediction leads to reduced development time, cost, reduced rework effort, increased customer satisfaction and more reliable software. Therefore, defect prediction practices are important to achieve software quality and to learn from past mistakes.

Keywords: Defect Prediction, Software Quality, Defect Detection, Bugs.

ISSN: 0976-5425
Vol-5-Issue-4-July-August-2019

INTRODUCTION:

As software testing is usually performed under the pressure to deliver the software, Software engineering community has focused on identification of faulty modules prior to the testing of software. Identification of faulty modules helps tester to pay more attention on testing the fault prone modules. Early detection of bugs helps to save cost and efforts. Different software metrics such as: software change matrices (SCM) and code based matrices (CBM) have been used by various researchers to identify the faulty modules in software. Quality of software cannot be achieved by identifying the bugs or errors only at the testing phase. Bugs are automatically injected at every phase of software development life cycle. So the detection of the bugs should be at every phase in place of only at the testing phase. High risk components within the software project should be caught as soon as possible, in order to enhance software quality. Software defects always increases the cost and time in completing a software product with expected quality. Moreover, identifying and rectifying defects is one of the most time consuming and expensive software processes. It is not practically possible to eliminate each and every defect but reducing the magnitude of defects

LITERATURE REVIEW:

Abdullah Alsaeedi et al (2019) An essential objective of software development is to locate and fix defects ahead of schedule that could be expected under diverse circumstances. Many software development activities are performed by individuals, which may lead to different software bugs over the development to occur, causing disappointments in the not-so-distant future.

Peng Hea et al (2014) Software defect prediction plays a crucial role in estimating the most defect-prone components of software, and a large number of studies have pursued improving prediction accuracy within a project or across projects. However, the rules for making an appropriate decision between within- and cross-project defect prediction when available historical data are insufficient remain unclear. The experimental results indicate that (1) the choice of training data for defect prediction should depend on the specific requirement of accuracy; (2) the predictor built with a simplified metric set works well and is very useful in case limited resources are supplied.

Jie Xu et al (2010) various statistical techniques and machine learning methods were used to variety the validity of software defect prediction models.. Here neuro fuzzy approach was used. Data from ISBSG were taken to carry out the work.

ISSN: 0976-5425
Vol-5-Issue-4-July-August-2019

Mohamad Mahdi Askari et al (2014) for the prediction of software defects used artificial neural network in order to better the generalization capability of the algorithm. Further support vector machine technique was used along with the learning algorithm and evolutionary technique. Thus this led to the maximization classification margin and prevented over fitting problem. This algorithm was tested with eleven machine learning models from NASA datasets. The conclusion drawn was that it provided better accuracy and precision than other models.

Catal et al (2010) investigated 90 software defect prediction, published between 1990 and 2009. He categorized these papers and reviewed each paper from the perspectives of metrics, learning algorithms, data sets, performance evaluation metrics, and experimental results in an easy and effective manner. Ac-cording to this survey, the author stated that most of the studies using method-level metrics and prediction models were mostly based on machine learning techniques, and Naıve Bayes was validated as a robust machine learning

OBJECTIVES:

1. To Study the high price-performance ratio of the prediction model built with a simplified subset of metrics in different scenario
2. To study the Software detect prediction
3. To perform efficient bug prediction in software, so that the cumbersome task of testing will become easier.

Software defect prediction: Software defect prediction is the process of locating defective modules in software. To produce high quality software, the final product should have as few defects as possible. Early detection of software defects could lead to reduced development costs and rework effort and more reliable software.

There are different types of classification for software defects, according to the Institute of Electrical and Electronics Engineers (IEEE) standard 104 which are:

1. Defect: The lack to perform the tasks and system requirements that are provided by developer and customers.
2. Error: This can be happening by customer due to the knowledge about the deficiency the software, produce incorrect results
3. Failure: Stop its previous required function of the software products or incorrect result will be provided for each input that is given by the customers.
4. Fault: it is clear or obvious error occurred in the software products.

ISSN: 0976-5425
Vol-5-Issue-4-July-August-2019

Depending on the above software defect classifications, a defect that is occurring in software product lead malfunctioning and health problems in case of serious and critical software, such as NASA software products for Space sciences.

Bug prediction: Bug prediction is needed to identify the different semantics and syntactical attributes analysis for identifying the error and bugs. Therefore it turns into a classification and pattern analysis problem. Additionally a lot of data is available in high dimensional attributes, thus it is needed to optimize it by enhancing classification accuracy and resource consumption optimization in terms of space and time complexity. The approach will find applicability in software development companies, to make the task of testing easier and to get a better, reliable and less faulty end product. It will prove to be helpful for software testers by making the test cases better and to the end user by delivering a better product.

For providing a solution to these problems, it is required to have better data sets. The huge dimensions can be reduced by dimension reduction techniques and outlier detection and removal can be done by using data mining and data clustering techniques. To avoid application of the method time and again, the system can be trained using a neural network, and regression analysis can be performed. The issue of cost will get resolved up to an extent by early prediction of bugs, rather than getting it accomplished at the testing phase.

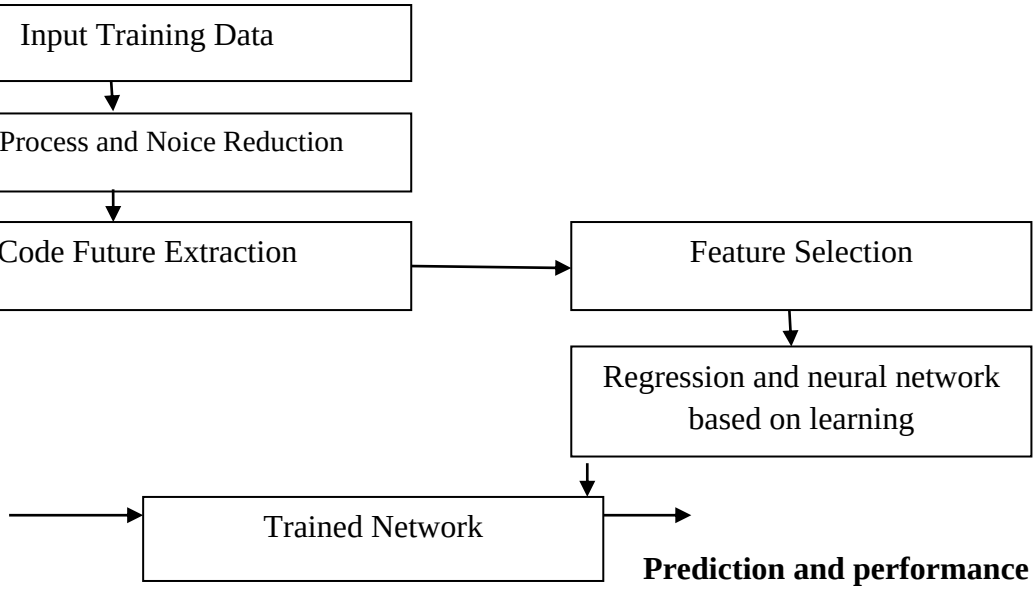

Figure: Steps of the proposed approach

ISSN: 0976-5425
Vol-5-Issue-4-July-August-2019

The proposed model helps in predicting the fault in the code changes. The main components of the presented system, mentioned in the flow diagram are:

1. Input training set: That is a simple user interface which accepts the raw training set. The training set may contains noisy data, and irrelevant instances.
2. Pre-process data set: In this phase the training data is transformed and normalized for finding the more relevant attributes and instances.
3. Code feature extraction: In this phase using neighbour component analysis method significant features are extracted.
4. Feature selection: Initially the calculated features are found in huge quantity therefore using K-PCA algorithm the dimensionality of the data is reduced for finding more appropriate features from the training samples.
5. Regression and neural network: Neural network is an essential data mining tool which can be implementable with the verity of applications such as recognition, pattern detection and other similar data models. But that suffers from the slow learning rate. Therefore the weight adjustment and initialization phase is modified using the regression analysis concept. That may improve the learning capability of the system**.**

Evaluation Measures for Software Bugs Prediction:

In this section, we will discuss different measurements for software defect prediction such as true positive (TP), true negative (TN), false positive (FP) and false negative (FN). TP denotes the number of defective software instances that are correctly classified as defective, while TN is the number of clean software instances that are correctly classified as clean. FP denotes the number of clean software instances that are wrongly classified as defective, and FN denotes the number of defective software instances that are mistakenly classified as clean.

One of the primary simple metrics to evaluate the performance of predictive models is classification accuracy, also called the correct classification rate. It is utilized to quantify the extent of the effectively classified instances to the aggregate instances. Another measure is called precision, and it is calculated by dividing the number of instances correctly classified as defective (TP) by the total number of instances classified as defective (TP + FP). In addition, recall measures the percentage of the number of instances correctly classified as defective (TP) to the total number of faulty instances (TP + FN) [16]. F-score is a harmonic mean of precision and recall, and many studies in the literature used F-score metrics. ROC-

ISSN: 0976-5425
Vol-5-Issue-4-July-August-2019

AUC calculates the area under the receiver operating characteristic (ROC) curve by computing trade-offs between TPR and FPR.

Accuracy = (TP+TN)/(TP+TN+FP+FN)

Precision = TP/(TP+FP)

Recall = TP/(TP+FN)

F-Score = (2 *Precision*Recall)/(Precision + Recall)

Classification of Defects:

A defect in software testing is an error in programming or logic that causes a program to failure or to produce wrong/unexpected outcome. Following are the different types of defects/bug: A. Invalid or rejected bug whenever scenario is wrong and developer does not accept it as a bug then it is called invalid/rejected bug. This bug generally occurred due to the following cases:

Case 1:- if Test Engineer misunderstood the requirement and may log the bug then developer will change the status to invalid. Then again Test Engineer go through it and if he also feel that it's invalid and closes the bug by keeping the status as close.

Case 2:- if developer misunderstood the requirement and changes the status to invalid after go through. The Test Engineer will go through the bug and he also may reproduce it and if the bug exists really he changes the status from invalid to re-open.

B. Duplicate bug when same bug is found by different Test Engineer then it is known as duplicate bug. This bug is occurred because of following reasons:

- Common features (modules) which is access by both the Test Engineer.
- Dependent features.

Solution for avoid the duplicate bug:

i. Search in bug repository, if bug already exists, do not log (report) a bug. If bug does not exist, log (report) a bug and store in bug repository.

ii. Send it to developer and carbon copy to all test engineers. C. Not-Reproducible bug Developer accepts the bug but not able to find the same bug after following the navigation steps mentioned in the bug report. Reasons for not-reproducible bug:

Platform mismatch

i. Server mismatch Test Engineer tests the application in one server and developer may reproduce the bug in another server, to avoid this mention server name in bug report.

ISSN: 0976-5425
Vol-5-Issue-4-July-August-2019

ii. Environment mismatch Test Engineer tests the application in different operating system and browser and developer may reproduce the bug in different operating system and browser, to avoid this mention platform name in bug report.

• Data mismatch Scenario may be correct, but for testing an application, Test Engineer use different data and developer may use different data for reproducing the bug, to avoid this mention test data in bug report.

• Build mismatch Test Engineer got a bug in one build and developer reproduce the same bug in different/another build due to time constraint it is known as build mismatch. Can't fix bug Developer accept the bug, also able to reproduce it but not able to perform code changes due to some reasons.

These reasons are as follows that's why developer can't fix the bug:

1. Core of code (bug is in the core of code)

2. No technology support If there will be major changes in bug then developer can't say that can't fix that bug. Can't fix bug should be minor bug but all minor bug cannot be can't fix bug.

E. In-consistent bug In first time Test Engineer found a bug but after that he is not able to find the same bug in next time, so to avoid the inconsistency Test Engineer should take the screenshot and follow the following steps:

1. As soon as Test Engineer got the bug first to take the final screenshot of that bug.

2. Re-confirm the bug whether it is consistent or not.

3. If bug is consistent then search the bug in bug repository. If bug report is not found then prepare the bug report and send it to the developer.

Defect prediction can offer an added probability to the event team to retest the modules or files that the faultiness chance is high. By spending longer on the defective modules and no time on the non-defective ones, the resources of the project would be used better and as a result, the upkeep part of the project are easier for each the customers and also the project owners. Their claim will be understood better once we notice that some outline defects as observed deficiencies whereas some others define them as residual ones. We have a tendency to explore the publications related to defect prediction we see that in early studies static code features were used a lot of.

CONCLUSION:

Software quality is the degree of conformance to explicit or implicit requirements and expectations. A software metric is a quantitative measure of a degree to which a software

ISSN: 0976-5425
Vol-5-Issue-4-July-August-2019

system or process possesses property with no defects. Hence, Software defect prediction model helps in early detection of defects using Classification Technique. In future we will be comparing the results of Supervised classification techniques on different datasets and open source projects to analyze the best classification technique to predict the defect in order to evolve a good software quality product. Further investigation need to be carried with emphasis on preprocessing of data and classifiers specifically designed for defect prediction in software modules.

REFERENCES:

1. Peng Hea, Bing Lic, Xiao Liua, Jun Chenb, Yutao Mab (2014), "An Empirical Study on Software Defect Prediction with a Simplified Metric Set",
2. Abdullah Alsaeedi, Mohammad Zubair Khan (2019), "Software Defect Prediction Using Supervised Machine Learning and Ensemble Techniques: A Comparative Study", Journal of Software Engineering and Applications, Volume: 12, Issue: 4, PP: 85-100
3. Jie Xu, Danny Ho and Luiz Fernando Capretz, "An Empirical Study On The Procedure Drive Software Quality Estimation Models", International journal of computer science & information Technology (IJCSIT) Vol.2, No.4, (2010).
4. Mohamad Mahdi Askari and Vahid Khatibi Bardsiri (2014), "Software Defect Prediction using a High Performance Neural Network", International Journal of Software Engineering and Its Applications Vol. 8, No. 12 (2014), pp. 177-188
5. Venkata U and R. A, "Empirical Assessment of Machine Learning based Software Defect Prediction Techniques " Proceedings of the 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems 2005.
6. Mikyeong P and E. H, "Software Fault Prediction Model using Clustering Algorithms Determining the Number of Clusters Automatically," International Journal of Software Engineering and Its Applications, vol. 8, pp. 199- 204, 2014.
7. Kalai Magal. R, Shomona Gracia Jacob. (2015). Improved Random Forest Algorithm for Software Defect Prediction through Data Mining Techniques, International Journal of Computer Applications (0975-8887), 117(23), 18-22.
8. Ye Xia, Guoying Yan, Qianran Si. (2013). A Study on the Significance of Software Metrics in Defect Prediction, Proceeding of Sixth International Symposium: Computational Intelligence and Design (ISCID) (pp.343-346).

ISSN: 0976-5425
Vol-5-Issue-4-July-August-2019

9. Ye Xia, Guoying Yan, Xingwei Jiang, Yanyan Yang. (2014). A New Metrics Selection Method for Software Defect Prediction, Proceeding of International Conference: Progress in Informatics & Computing (pp.433-436)
10. Norman Fenton, Paul Krause and Martin Neil, (1999), "A Probabilistic Model for Software Defect Prediction", For submission to IEEE Transactions in Software Engineering
11. Jie Xu, Danny Ho and Luiz Fernando Capretz, "An Empirical Study On The Procedure Drive Software Quality Estimation Models", International journal of computer science & information Technology (IJCSIT) Vol.2, No.4, (2010)
12. Mohamad Mahdi Askari and Vahid Khatibi Bardsiri (2014), "Software Defect Prediction using a High Performance Neural Network", International Journal of Software Engineering and Its Applications Vol. 8, No. 12 (2014), pp. 177-188