Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 46 (2015) 906 – 912

Procedia Computer Science

International Conference on Information and Communication Technologies (ICICT 2014)

Open Issues in Software Defect Prediction

Ishani Arora[a], Vivek Tetarwal[a,]*, Anju Saha[a]

[a]*University School of Information and Communication Technology*
Guru Gobind Singh Indraprastha University, Dwarka, Delhi- 110078, India

Abstract

Software Defect Prediction (SDP) is one of the most assisting activities of the Testing Phase of SDLC. It identifies the modules that are defect prone and require extensive testing. This way, the testing resources can be used efficiently without violating the constraints. Though SDP is very helpful in testing, it's not always easy to predict the defective modules. There are various issues that hinder the smooth performance as well as use of the Defect Prediction models. In this report, we have distinguished some of the major issues of SDP and studied what has been done so far to address them.

Peer-review under responsibility of organizing committee of the International Conference on Information and Communication Technologies (ICICT 2014)

*Keywords:*data mining;defect prediction;machine learning;software quality;software testing

1. Introduction

Defect Prediction in software is viewed as one of the most useful and cost efficient operation. Software practitioners see it as a vital phase on which the quality of the product being developed depends. It has taken up major part in bringing down the allegations on the software industry, of being incapable to deliver the requirements within budget and on time. Besides this, the clients' response regarding the product quality has shown a large shift from unsatisfactory to satisfactory.

Today, many data miners have replaced the earlier statistical approaches for defect prediction. The basis of data mining is the classification model which places the component in one of the two classes: fault prone or non fault prone. Initially, already known instances, whose class we already recognize, are supplied to the classifier. Once the

* Corresponding author. Tel.: +918826658489.
*E-mail address:*tetarwalvivek1991@gmail.com

Peer-review under responsibility of organizing committee of the International Conference on Information and Communication Technologies (ICICT 2014)
doi:10.1016/j.procs.2015.02.161

model is trained, this is tested on the unknown instances and prediction performance of the technique is judged. Most of the study has been performed using the requirement[1, 3] and design metrics[1, 2].

Numerous amount of work has been done in the field of software defect prediction involving a varied number of techniques such as bagging[4, 5], boosting[4, 5], naive bayes[1, 5, 6], one rule[1, 6, 7], support vector machine[8, 9, 10], J48 decision tree[1, 6, 9], etc. A brief review of the significant work carried out in this field has been provided by Arora and Saha[11].

Researchers have identified a bunch of problems in this area of software defect prediction and have tried to offer the answers as well. However, the proposed solutions have not been accepted universally and are still left unanswered. This paper identifies and provides a brief overview of such issues. Nevertheless, this is not a complete view of the total work done so far in the area of software fault prediction.

Section II presents the issues which are still unresolved in this area and a brief overview of the work carried out in the direction of solving these issues; and the section following gives the conclusions as well as future directions to the scholars and researchers.

2. Issues in Software Defect Prediction

This section introduces the problems faced in software defect prediction and the solutions proposed by the eminent researchers for these problems. It also addresses the unresolved issues in this area as well.

2.1. Relationship between Attributes and Fault

Researchers are not capable to identify a generalized subset of attributes which act as a substantial factor for a module to be incorrect or non faulty. Also, there has been a controversy between which level of metrics to be applied, requirement metrics, design metrics or source code metrics, for analysis purpose. A variety of studies[1, 2, 3, 6, 12] have been performed, but unfortunately, with a different set of attributes every time. Overviews of the studies which have tried to address the above issue have been described below.

Emam et al.[2] examined the set of design metrics which act as a strong factor in determining the occurrence of the bug. Object oriented features were included in this study. The analysis was done using area under ROC curve (AUC) as an evaluation criterion. The results concluded that inheritance and export coupling (EC) strongly associated with fault proneness.

Another similar experiment was performed by Koru and Liu[12] where they compared the method level as well as class level metrics to forecast the defect prone modules in five datasets of NASA[13]. The decision tree based and instance based classifiers were applied in WEKA[14] for building the defect prediction models. These models were evaluated using precision, recall and F- measure. The results concluded that one should prefer using larger components for fault prediction instead of smaller ones. They also suggested using class level metrics for defect predictions involving smaller size modules.

Jiang et al.[1] verified that the attributes available to the software developers can aid them to identify defects as early as possible in the software lifecycle. They performed analysis, which compared the defect prediction models developed from the requirement metrics, source code metrics and combination of requirement and code metrics. The machine learning algorithms included one rule learner, naive bayes, voted perceptron, logistic regression, J48 decision tree and random forests. The results reported that the textual metrics combined with static code metrics improve the defect predictors except, in the case of voted perceptron.

Sandhu et al.[3] explored the applicability of requirement metrics by quantitatively comparing them with the code based metrics and a combination of both of them. First, k-means clustering is applied in the transformation phase and then the decision tree learner on CM1 project of MDP[13] public repository. Optimal solutions are achieved with 100 percent precision and 100 percent recall. Hence, it is useful to make the prediction model with the metrics available in the early phases of the software lifecycle.

In contrast to all the above studies, Menzies et al.[6] presented a significant statement that the classification methods used to build the defect prediction models are more important and not the metrics that are involved in the experiment. They examined eight data sets of NASA MDP[13] using decision tree based, rule based and bayes theorem based learners. The experiment also compared the results of no filter with log filter. Recall, balance and the probability of not having the false alarm were included for evaluation. The results proved that naive bayes with log

filter outperforms the OneR and J48 defect prediction models. They also proved that the naive bayes with log filter have an average Pd of 71 percent and Pf of 25 percent.

As discussed in the above studies, none of them has been able to establish a generalized association between the attributes and fault.

2.2. No Standard Measures for Performance Assessment

A great deal of inconsistency in choosing the performance reporting measures has been experienced across several subject areas. No standard criterion has been set for analyzing and comparing the defect prediction models. A brief description of research carried out by some eminent researchers in this direction is presented below.

Menzies et al. preferred reporting recall and Pf for evaluation purpose in[6]. However, Zhang and Zhang[15] argued to report recall and precision instead of recall and Pf. In reply, Menzies et al.[16] explained the problem of precision instability and that's why they suggested it should not be used for operational assessment.

In cross project defect prediction, Gray et al.[17] suggested that it should be in practice to report precision also, if the data suffers from the class imbalance problem (described in section *2.6*). They used data sets of thirteen projects of NASA[13] repository. The authors evaluated the classification techniques on the basis of recall, chances of having the false alarms and precision. They said that recall and precision must be covered in case of class imbalance problem.

In another experiment, Jiang et al.[5] claimed that variance is an important criterion for evaluation of software defect predictors. They statistically studied the importance of variance on twelve data sets from NASA MDP[13] repository. The criteria used for performance assessment are recall, f-measure, precision and the area under ROC curves (AUC). The methods employed to learn the prediction models include random forest, bagging, logistic regression, boosting and naive bayes. The outcome reported that AUC has lower variance, therefore it is more static. The authors reported that the model with the lowest variance is highly preferable.

Yet, in all the studies identified above, it is suggested to report recall in combination with other criteria, but, no standard combination has been offered so far.

2.3. Issues with Cross-Project Defect Prediction

It is generally preferred for a model to learn using the locally available data, which is usually very similar to the data on which it is to be tested. This local data can be taken from some previous versions of the same project, or from some other similar project using the same programming language. Nevertheless, most of the times the risk management team of an organization faces the problem of unavailability of this local site information. The unavailability of training data may be due to many reasons, such as no similar project has been previously developed or the current technology has changed.

To resolve this problem, researchers came up with a solution of cross project defect prediction, where the defect prediction model is developed on one project and examined on some other. These two projects could be same or completely different. Unfortunately, the performance of the models built using cross company data has not been very promising.

Turhan et al.[18] compared the performance of models built using cross company data and localized data for defect prediction on ten projects of PROMISE[19] repository and concluded that the localized predictors are preferable over cross company ones. But, in case of unavailability of localized data the only viable solution is the use of cross-company data. Many researchers have worked in this area proposing the methods to use cross project data efficiently for defect prediction. Some of these studies are discussed below.

Watanabe et al.[20] identified the issue of unavailability of training data for defect predictors. They performed intra and inter project analysis using object oriented metrics using Sakura editor and jEdit projects. The evaluation uses precision and recall as the performance measures. The results showed that inter project defect prediction is useful between the similar projects, but the effects are not really acceptable.

Peters et al.[8] applied CLIFF and MORPH techniques together to maintain the privatization as well as utilization of the organizational data. The analysis was done on data sets of ten object oriented projects taken from the PROMISE[21] repository. The three classification algorithms - naive bayes, neural networks and support vector

machine, implemented in WEKA[14] were used. The method proposed by them was assessed on the basis of recall, g – measure and probability of false alarm. The results showed that this approach made no compromise in predicting defects while sharing the private data.

Peters et al.[22] presented peters filter for filtering out irrelevant data from the entire cross project data. The useful data was then used during the learning phase of the predictive model. The learnt classifier was then used for fault prediction at the local site. The peters filter was compared with the Burak filter[18] on the basis of accuracy, recall, false alarm rate, f-measure and g-measure using class level metrics. The analysis was done with 56 data sets from PROMISE[21] repository, out of which 35 were used for training purpose and 21 data sets were used during model testing. The classifiers that were used include random forest, naive bayes, logistic regression and nearest neighbor (k=1) algorithm. It was concluded that the small data sets were not suitable for local site prediction and the proposed peters filter was sixty four percent higher in performance than Burak filter[18].

Herbold[9] proposed two methods for improving the results of inter project defect prediction. These machine learning algorithms were based on the distance between the instances such as expectation-maximization (EM) clustering and nearest neighbor selection. The experiments were performed on 44 data sets from 14 open source projects. Logistic regression, naive bayes, bayesian networks, support vector machine, J48 decision tree, random forest and multi layer perceptron models were applied. The results proved that both the proposed methods were highly preferable in enhancing the operation of cross project defect predictors. Nevertheless, the local site prediction still outperforms the cross project fault prediction.

As discussed above, the proposed methods have improved the efficiency/performance of models built using cross project data, yet there is great scope of improvement in this area.

2.4. No General Framework Available

Another issue with this upcoming field is that different scholars and researchers have used different techniques on different data sets. But, there has been no standard framework or procedure to apply a software defect prediction process on a local or cross company project. A few studies have proposed different frameworks for the implementation of software defect prediction models, which are discussed below.

Song et al.[7] proposed a framework which compared the predictors comprehensively and in an unbiased manner. The applicability of the framework is evaluated using simulated data and data available from the public repositories. The learning schemes are naive bayes, J48 decision tree and one rule learner. These schemes are operated in combination with data pre-processors and attribute selection methods. The results are compared on the basis of recall, area under ROC (AUC) and balance measures. The authors reported that this framework is more efficient and the learning scheme of the model depends on the type of the data from which the model will be trained.

Chen et al.[10] proposed a semi-automated method for improving the software development process. Nine software projects are employed and a comparative analysis has been done between the proposed method and the other standard methods. The classification techniques involved decision trees, naive bayes, support vector machines, Adaboost and logistic regression. The judgment criteria were precision, recall, f-measure, accurately, area under ROC curve and root mean squared error. A new measure “process execution qualification rate” was proposed and also used for evaluation. The results have proven that the proposed approach is advantageous as compared to the other classifiers.

Although the proposed approaches have proven to be advantageous, but are very different from each other and are not generalized.

2.5. Economics of Software Defect Prediction

The irony of the discipline of software defect prediction is that most of the work has been done considering its ease of use and very few of them have focused on its economical position. The misclassification can prove to be real pricey, particularly in the case of predicting faulty component as non faulty. Hence, determining the answer to when and how much utility it has is very important. A few studies answering these inquiries have been discussed below.

Jiang et al.[4] investigated this area of software defect prediction with a view of analyzing the cost involved in misclassifying the units/modules of the system. The study is done on sixteen projects from MDP[13] and PROMISE[20]

public repositories. The classification methods that were applied are: random forest, boosting, logistic regression, naive bayes and bagging. It was graphically shown that the cost should be restrained in mind as a significant factor during the assessment of the fault prediction models.

Banthia et al.[23] proposed a cost evaluation framework to actually evaluate whether the fault prediction is useful and if yes, when it is beneficial to use. The study was performed on 19 data sets of MDP[13] and PROMISE[20] repository and applied five algorithms: random forests, J48 decision tree, k-means clustering, neural network and instance based classifier. Two cases, with and without fault prediction, were compared using normalized estimated cost. The results concluded that software defect prediction is beneficial only when the project under study has the number of faulty modules in the range of 21 to 42 percent. Also, random forests perform better when the faulty modules are less and J48 and neural network is better in case of higher number of faulty modules. However, none of the techniques superseded the other in all the cases.

The above two studies have considered the economical aspect of software defect prediction models, yet some more exploration is needed in this direction.

2.6. Class Imbalance Problem

The efficiency of Software Fault prediction models is greatly shaped by the class distribution of the training data[24]. Class distribution is described as the number of instances of each class in the training dataset. If the number of instances belonging to one class is much more than the number of instances belonging to another class, then the problem is known as class imbalance problem[25]. The class with more instances is called majority class and the one with lesser instances is called minority class. The problem widens when the class under consideration, i.e. the faulty class is represented by fewer instances. Various techniques have been proposed for addressing this problem and a few are discussed below.

Barandela et al.[26] carried out a comparative study of various sampling, i.e. resizing techniques including Undersampling and Oversampling techniques for handling class imbalance and analyzed the relative performance of these two categories of sampling techniques. They concluded that, in case of highly imbalanced data sets, oversampling of minority class should be done, whereas, if the datasets are not severely biased then undersampling is better. Likewise, the combination of Undersampling and Oversampling can be a safer alternative.

An empirical study was carried out by Zhou and Liu[27] for studying the effect of sampling and threshold moving on the training of cost-sensitive neural networks. The results showed that cost sensitive learning is easy for binary class dataset and difficult for multi-class dataset as well as for a highly imbalanced dataset.

The effect of sampling rate on the performance of defect prediction models was carried out by Pelayo and Dick[28] using 4 NASA datasets. The sampling technique used was Synthetic Minority Oversampling Technique (SMOTE) and the classifier used was Decision Tree Classifier.

Khoshgoftaar et al.[29] compared 5 data sampling techniques, namely, Random Under Sampling (RUS), SMOTE, Borderline-SMOTE, Random Over Sampling (ROS) and Wilson's Editing (WE) with Boosting algorithm and concluded that sampling performs significantly well but Boosting performs even better.

A Genetic Algorithm based sampling technique called Evolutionary Sampling was proposed by Khoshgoftaar et al.[30] and compared with various imbalance handling methods using two classifiers namely C4.5 and RIPPER. The proposed technique proved to be better than the other techniques.

Khoshgoftaar et al.[31] offered a hybrid Sampling/Boosting algorithm to address class imbalance namely RUSBoost. The performance of the proposed algorithm was then compared to its individual techniques, namely RUS and Boosting as well as with another hybrid Sampling /Boosting algorithm SMOTEBoost and its individual techniques i.e. SMOTE. It was concluded that, the operation of two hybrid algorithms is not significantly different, but RUSBoost is simpler and quicker than the other.

Galar et al.[25] proposed a taxonomy of ensemble based algorithms for addressing class imbalance. They also showed empirically that ensemble methods improve the performance of prediction models as compared to preprocessing techniques applied to a single classifier model.

Wang and Yao[32] investigated if class imbalance learning can benefit software defect prediction and if yes, then how. They compared various imbalance learning methods, namely RUS, Balanced RUS, threshold moving, SMOTEBoost and AdaBoost. NC and observed that the best performance wass achieved using AdaBoost. NC.

The impact of class noise along with class imbalance on the performance of classifier algorithms was studied by Seiffert et al.[33]. They found that class noise impacts the performance more significantly and as the level of class noise in the data set decreases, the performance of the classifier improves.

Tetarwal and Saha[34] combined cost sensitive ensembles with preprocessing techniques and concluded that all the combinations as well as the best individual technique was outperformed by Cost Sensitive Boosting with Resampling.

Although various techniques have been proposed by various researchers to address this issue, but not a single technique outperformed the others in all the studies. Hence we cannot reach a conclusion as to which technique should be applied to solve the class imbalance problem.

3. Conclusions and Future Directions

Software defect prediction is seen as the phase of enhancing the software quality. It helps us to forecast the future, i.e. to identify the modules which are likely to have faults. This aids the software project management team to deal with those areas in the project on a timely basis and with sufficient effort. This research area has emerged since 1990s. With nearly 24 years of its history, this area still lacks in solving some issues. This paper has shown and analyzed what has been done so far and what needs to be done ahead. An aggregate of six problems was discussed: finding the set of attributes to be correlated with fault, the absence of standard measures for performance assessment, problems with cross project defect prediction, economics of software defect prediction and class imbalance problem and the absence of any general framework for the software defect prediction.

For finding relationship between attributes and faults, more studies can be carried out using all the combinations and subsets of attributes with various classifiers. Also, the studies may verify if there is a relationship between attributes and faults, or not. For assessing performance of a prediction model, the importance of each measure can be found out and depending on their importance, a new measure can be proposed by weighted combination of all the measures. In case of cross project defect prediction, prior to training the model using cross-project defect data, the data should be refined using some pre-processing method so that it becomes similar to the localized data. In order to provide a general framework, one may consider the effect of classifier combination, order of training data, effect of pre-processing techniques and attribute selection on the performance of prediction model. For the purpose of economic assessment of application of a defect prediction model, the estimated cost of defect prediction using prediction model along with misclassification cost may be taken into account and compared with the estimated cost of defect prediction without using prediction models. For this, a model needs to be proposed for assessing these costs. In order to address the class imbalance problem in Software defect prediction, more focus should be on providing algorithmic alternatives as preprocessing techniques affect the actual information of the dataset. From the issues discussed in this paper, one can get a better understanding of the problems still prevailing in the field of software defect prediction and affecting the applicability and effectiveness of the software defect prediction models. Thus, from the issues stated in this paper, the researchers may get directions to carry out studies oriented towards finding generalized solutions for the problems.

References

1. Jiang Y, Cukic B, Menzies T. Fault prediction using early lifecycle data. In: *18th IEEE International Symposium on Software Reliability*. Trollhattan; 2007. p. 237-246.
2. Emam KE, Melo W, Machado JC. The prediction of faulty classes using object-oriented design metrics. *Journal of Systems and Software* 2001; **56**:63-75.
3. Sandhu PS, Brar AS, Goel R, Kaur J, Anand S. A model for early prediction of faults in software systems. In: *2nd International Conference on Computer and Automation Engineering*. Singapore; 2010. p. 281-285.
4. Jiang Y, Cukic B, Menzies T. Cost curve evaluation of fault prediction models. In: *19th International Symposium on Software Reliability Engineering.* Seattle; 2008. p. 197-206.
5. Jiang Y, Lin J, Cukic B, Menzies, T. Variance analysis in software fault prediction models. In: *20th International Symposium on Software Reliability Engineering*. Mysuru; 2009. p. 99-108.
6. Menzies T, Greenwald J, Frank A. Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering* 2007; **33**:2-13.

7. Song Q, Jia Z, Shepperd M, Ying S, Lin J. A general software defect-proneness prediction framework. *IEEE Transactions on Software Engineering* 2011; **37**:356-70.
8. Peters F, Menzies T, Gong L, Zhang H. Balancing privacy and utility in cross-company defect prediction. *IEEE Transactions on Software Engineering* 2013; **39**:1054-68.
9. Herbold S. Training data selection for cross-project defect prediction. In: *9th International Conference on PROMISE '13*. Baltimore; 2013. p. 1-10.
10. Chen N, Hoi SCH, Xiao X. Software process evaluation: A machine learning framework with application to defect management process. *Empirical Software Engineering* 2013; **19**:1531-64.
11. Arora I, Saha A. A literature review on software defect prediction. In: *Second International Conference on Emerging Research in Computing, Information, Communication and Applications*. Bangalore; 2014. p. 478-487.
12. Koru AG, Liu H. Building effective defect-prediction models in practice. *IEEE Software* 2005; **22**:23-9.
13. http://mdp.ivv.nasa.gov.
14. http://www.cs.waikato.ac.nz/ml/weka/.
15. Zhang H, Zhang X. Comments on "Data mining static code attributes to learn defect predictors". *IEEE Transactions on Software Engineering* 2007; **33**:635-37.
16. Menzies T, Dekhtyar A, Distefano J, Greenwald J. Problems with precision: A response to "Comments on 'Data mining static code attributes to learn defect predictors'". *IEEE Transactions on Software Engineering* 2007; **33**:637-40.
17. Gray D, Bowes D, Davey N, Sun Y, Christianson B. Further thoughts on precision. In: *15th Annual Conference on Evaluation & Assessment in Software Engineering*. Durham; 2011. p. 129-133.
18. Turhan B, Menzies T, Bener AB, Stefano JD. On the relative value of cross-company and within-company data for defect prediction. *Empirical Software Engineering* 2009; **14**:540-78.
19. Boetticher G, Menzies T, Ostrand T. Promise repository of empirical software engineering data. 2007. Available at: *http://promisedata.org/*.
20. Watanabe S, Kaiya H, Kaijiri K. Adapting a fault prediction model to allow inter language reuse. In: *4th International Workshop on PROMISE '08*. Leipzig; 2008. p. 19-24.
21. Menzies T, Caglayan B, Kocaguneli E, Krall J, Peters F, Turhan B. The promise repository of empirical software engineering data. 2012. Available at: *promisedata.googlecode.com.*
22. Peters F, Menzies T, Marcus A. Better cross company defect prediction. In: *10th IEEE Working Conference on Mining Software Repositories*. San Francisco; 2013. p. 409-418.
23. Bathia D, Gupta A. A framework to assess the effectiveness of fault-prediction techniques for quality assurance. In: *7th CSI International Conference on Software Engineering*. Pune; 2013. p. 40-49.
24. Batista GEAPA, Prati RC, Monard MC. A study of the behavior of several methods for balancing machine learning training data. *SIGKDD Explorations* 2004; **6**:20-9.
25. Galar M, Fernandez A, Barrenechea E, Bustince H, Herrera F. A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews* 2012; **42**:463-84.
26. Barandela R, Valdovinos RM, Sánchez JS, Ferri FJ. The imbalanced training sample problem: Under or over sampling? *Structural, Syntactic, and Statistical Pattern Recognition*. Springer Berlin Heidelberg; 2004. p. 806-814.
27. Zhou ZH, Liu XY. Training cost-sensitive neural networks with methods addressing the class imbalance problem. *IEEE Transactions on Knowledge and Data Engineering* 2006; **18**:63-77.
28. Pelayo L, Dick S. Applying novel resampling strategies to software defect prediction. In: *Annual Meeting of the North American Fuzzy Information Processing Society (NAFIPS'07)*. 2007. p. 69-72.
29. Seiffert C, Khoshgoftaar TM, Van Hulse J, Napolitano A. Building useful models from imbalanced data with sampling and boosting. In: *FLAIRS Conference*. 2008. p. 306-311.
30. Drown DJ, Khoshgoftaar TM, Seliya N. Evolutionary sampling and software quality modeling of high-assurance systems. *IEEE Trans. Syst., Man, Cybern, Part A: Systems and Humans* 2009; **39**:1097–107.
31. Seiffert C, Khoshgoftaar TM, Van Hulse J, Napolitano A. RUSBoost: A hybrid approach to alleviating class imbalance. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans* 2010; **40**:185-97.
32. Wang S, Minku LL, Yao X. A learning framework for online class imbalance learning. In: *IEEE Symposium on Computational Intelligence and Ensemble Learning (CIEL)*. 2013. p. 36-45.
33. Seiffert C, Khoshgoftaar TM, Van Hulse J, Folleco A. An empirical study of the classification performance of learners on imbalanced and noisy software quality data. *Information Sciences* 2014; **259**:571-95.
34. Tetarwal V, Saha A. Combining cost-sensitive ensembles with pre-processing techniques for addressing class-imbalance. In: *International Conference on Communication and Computing ICC-2014*. Bangalore; 2014. p. 565-576.