IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 8, Issue 12, December 2019

A SURVEY ON SOFTWARE DEFECT DETECTION

Y.Vanaja, V.B.Buvaneswari

M.Phil. Research Scholar, PG & Research Department of Computer Science, Government Arts College Coimbatore, India

Assistant Professor,PG & Research Department of Computer Science, Government Arts College Coimbatore, India

Abstract: Software defect prediction plays an important role in improving software quality and reducing time and cost for software testing. It is one of the most demanding activities of the testing phase of system development life cycle (SDLC). It identifies defect prone modules that require extensive testing and utilizes testing resources effectively. In large software projects, prediction techniques will play a crucial role in aiding developers for speedy marketing of reliable software products. This survey introduces the common defect prediction process used in the literature, discusses methods to evaluate defect prediction performance and compares various defect prediction techniques that includes metrics, models, and algorithms. It also deals applications involving defect prediction and other emerging topics. Finally, this survey assists to identify challenging issues that creep up during software defect prediction.

Keywords: Software defect, Clustering, Classification, Prediction

I. INTRODUCTION

Software Defect (Bug) Prediction is one of the most active research areas in software fields. 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 with defect prediction models provide list of bug-prone software artifacts and quality assurance teams have to effectively allocate limitedly available resources for testing software products. They introduce the common software defect prediction process and several research streams in defect prediction. Since different evaluation measures for defect prediction has been used across the literature, they present different evaluation measures for defect prediction models. It includes defect prediction metrics and prediction granularity. Before building defect prediction models, some studies applied preprocessing techniques to improve prediction performance. This paper briefly investigate preprocessing techniques used. Recent defect prediction studies have focus on cross-project defect prediction and various cross-project defect prediction approaches are compared. This discussion is about applications using defect prediction results which is an emerging topic. This survey raises challenging issues for defect prediction.

II. LITERATURE REVIEW

Defect prediction studies are still having many challenging issues. Even though there are many outstanding studies, it is not easy to apply those approaches in practice because most studies that are verified in open source software projects may not work for any other software products including commercial software. However, proprietary datasets are not publicly available because of privacy issues.

- Although Peter et al. proposed MORPH algorithm to increase data privacy, MORPH was not validated in cross-project defect prediction. Investigating privacy issue in cross project defect prediction is required if they have more available proprietary datasets.
- Arisholm E., Briand., L. C., and Fuglerud M. discussed that Cross prediction is still a very difficult problem in defect prediction in terms of two aspects. Different feature space: There are many publicly available defect datasets. However, they cannot use many of datasets for cross prediction since datasets from different domains have different number of metrics (features). Prediction models based on machine learning cannot be built on the datasets, which have different feature spaces. Feasibility: Studies on cross prediction feasibility are not mature yet. Finding general approaches to check the feasibility in advance will be very helpful for practical use of cross prediction models.

IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319-5940
International Journal of Advanced Research in Computer and Communication Engineering
Vol. 8, Issue 12, December 2019

- Ambrose M. D., Lanza M., and Robbes R. discussed that if software projects are getting larger, file-level defect prediction may not be enough in terms of cost-effectiveness. There are still few studies for finer prediction granularity. Studies on finer-grained defect prediction such as line-level defect prediction and change classification are required.
- Bacchelli, M. D'Ambros, and Lanza M. discussed that defect prediction metrics and models proposed until now may not always guarantee generally good prediction performance. As software repositories evolve, they can extract new types of development process information, which never used defect prediction metrics/models. New metrics and models need to be kept investigating.
-

III. OVERVIEW OF SOFTWARE DEFECT PREDICTION:

a) Software defect prediction process

Software defect prediction is the process of locating defective modules in software. It helps to improve software quality and testing efficiency by constructing predictive models from code attributes to enable a timely identification of fault-prone modules. It also helps us in planning, prediction, monitoring and control but needs improvement in terms of cost-effectiveness and granularity. Studies on line-level defect prediction and change classification are required.

- Defect prediction metrics and models does not always give good prediction performance. It needs new metrics and models.
- Prediction process density and to better understand and control the software quality. The Software Defect Prediction result is the number of defects remaining in a software system that can be used to control the software process. Data Mining is most popular as a business information management tool that is expected to reveal knowledge structures that can guide in conditions of limited certainty. Recently, there has been increased interest in developing new analytic techniques specifically designed to address the issues relevant to business Data Mining (e.g., Classification Trees), but Data Mining is still based on the conceptual principles of statistics including the traditional Exploratory Data Analysis (EDA) and modeling. An important general difference in the focus and purpose between Data Mining and the traditional Exploratory Data Analysis (EDA) is that Data Mining is more oriented towards applications.

a) Data Mining is relatively less concerned with identifying the specific relations between the involved variables. For example, uncovering the nature of the underlying functions or the specific types of interactive, multivariate dependencies between variables are not the main goal of Data Mining. Instead, the focus is on producing a solution that can generate useful predictions.

b) Data Mining accepts a "black box" approach to data exploration or knowledge discovery and uses not only the traditional Exploratory Data Analysis (EDA) techniques, but also such techniques as Neural Network**s** which can generate valid predictions but are not capable of identifying the specific nature of the interrelations between the variables on which the predictions are based. The accuracy of the system is from 80% to 92% based on the algorithms.

Fig. 1: Common process of software defect prediction

Figure 1. The prediction model can predict whether a new instance has a bug or not. The prediction for bug-proneness (buggy or clean) of an instance stands for binary classification, while that for the number of bugs in an instance stands for regression.

3IV.BENEFITS AND DRAWBACKS OF SOFTWARE DEFECT

Apart from the representative papers discussed in previous sections, there are interesting and emerging topics in defect prediction study. One topic is about defect data privacy and the other topic is the comparative study between defect prediction models and static bug finders.

A. Defect Data Privacy

Peters et al. proposed MORPH that mutates defect datasets to resolve privacy issues in defect datasets. To accelerate cross-project defect prediction study, publicly available defect datasets are necessary. However, software companies are reluctant to share their defect datasets because of "sensitive attribute value disclosure". Thus, cross-project defect prediction studies usually conducted on open source software products or very limited proprietary systems. Experiments conducted by Zimmermann et al. for cross-project defect prediction are not reproducible since Microsoft defect datasets are not publicly available. To address this issue, MORPH moves instances in a random distance by still keeping class decision boundary. In

IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319-5940
International Journal of Advanced Research in Computer and Communication Engineering
Vol. 8, Issue 12, December 2019

this way, MORPH could privatize original datasets and still achieve good prediction performance as in models trained by original defect datasets.

B. Comparing Defect Prediction Models to Static Bug Finders

In contrast to defect prediction models (DP), static bug finders (SBF) detect bugs by using "semantic abstractions of source code". Rahman et al. compared defect prediction techniques and static bug finders in terms of cost-effectiveness. Rahman et al. found that DP and SBF could compensate each other since they may find different defects. In addition, SBF warnings prioritized by DP could lead to better performance than SBF's native priorities of warnings. This comparative study provided meaningful insights that explains how different research streams having the same goal can be converged together to achieve the better prediction/detection of defects.

Automatic identification of software faults has enormous practical significance. It requires characterizing program execution behavior and the use of appropriate data mining techniques on the chosen representation. In this paper the sequence of system calls are used to characterize program execution. The data mining tasks addressed are learning to map system call streams to fault labels and automatic identification of fault causes. Spectrum kernels and SVM are used for the former while latent semantic analysis is used for the latter. In this paper techniques are demonstrated for the intrusion dataset containing system call traces. The results show that kernel techniques are as accurate as the best available results but are faster by orders of magnitude. This paper aims to show that latent semantic indexing is capable of revealing fault-specific features

C. Categories of Software Defect Prediction Studies

A large number of research studies in the past decade have focused on proposing new metrics to build prediction models. Extensively studied metrics are source code and process metrics. Source code metrics measure how source code is intricate and the main rationale of the source code metrics is that source code with higher complexity can be more bug-prone. Process metrics are extracted from software archives such as version control systems and issue tracking systems that manage all development history. Process metrics quantify many aspects of software development process such as changes of source code, ownership of source code files, developer reciprocal actions, etc. Usefulness of process metrics for defect prediction has been proved in large numbers of studies. Most defect prediction studies are organized and based on statistical approach, i.e. machine learning. Prediction models learn by machine learning algorithms can predict either bug-proneness of source code (classification) or the number of defects in source code (regression).

Many research studies adopted recent machine learning techniques such as active/semi-supervised learning to improve prediction performance. Researchers also have proposed approaches for cross-project filth prediction. Most representative studies described above have been organized and verified under the within-prediction setting, i.e. prediction models were context and tested in the same project. However, it is toilsome for new projects, which do not have enough development historical information, to build prediction models. Representative approaches for cross defect prediction are metric compensation, Nearest Neighbor (NN) Filter, Transfer Naive Bayes (TNB), and TCA. These approaches rectify a prediction model by selecting similar instances, transforming data values, or developing a new model. Another interesting topic in cross-project defect prediction is to interrogate feasibility of cross-prediction. Many studies confirmed that cross-prediction is hard to achieve; only many cross-prediction combinations work. Identifying cross-prediction feasibility will play a vital role for cross-project filth prediction. There is a couple of studies respecting cross prediction feasibility based on decision trees. However their decision trees were verified only in special software datasets and were not deeply investigated.

V. EVALUATION MEASURES

A. Measures for Classification

To measure defect prediction results by classification models, they should consider the following prediction outcomes first:

- True positive (TP): buggy instances predicted as buggy.
- False positives (FP): clean instances predicted as buggy.
- True negative (TN): clean instances predicted as clean.
- False negative (FN): buggy instances predicted as clean.

With these outcomes, they can define the following measures, which are mostly used in the defect prediction literature.

a) **False positive rate (FPR)**: False positive rate is also known as the probability of false alarm (PF). PF measures how many tidy instances are predicted as buggy among all clean instances.

FPR= (FP) / (TN + FP)

b) Accuracy:

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

Accuracy refer both true positives and true negatives over all instances. In other words, accuracy shows the ratio of all accurately classified instances. However, accuracy is not proper measure particularly in filth prediction because of class

IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319-5940
International Journal of Advanced Research in Computer and Communication Engineering
Vol. 8, Issue 12, December 2019

imbalance of filth prediction datasets. For example, the average buggy rate of PROMISE datasets used by Peter et al. is 18%. If assume a prediction model that predicts all instances as clean, the accuracy will be 0.82 although no buggy instances are correctly predicted. This does not make sense in terms of filth prediction performance.

c) Precision:

PRE= (TP)/ (TP + FP)

d) Recall: Recall is also known as probability of detection (PD) or true positive rate (TPR) .Recall measures correctly predicted buggy instances among all buggy instances.

RE= TP/ TP + FN

e) F-measure: F-measure is a harmonic mean of precision and recall

F – M = 2 * (Precision * Recall) / (Precision + Recall)

Since precision and recall have trade-off, f-measure have been used in many papers.

B. Measures for Regression:

To measure defect prediction results from regression models, measures based on correlation calculation between the number of actual bugs and predicted bugs of instances have been used in many defect prediction papers .The representative measures are Spearman's correlation, Pearson correlation, R2 and their variations. These measures also has been used for correlation analysis between metric values and the number of bugs.

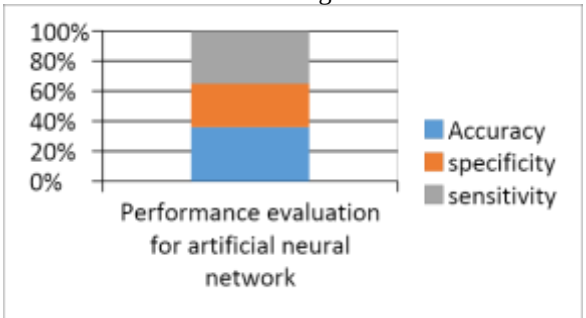

Fig: 2 Evaluation of Measures Regression

Figure 2 shows the count of evaluation measures used in the representative defect prediction papers for classification. As shown in the figure, f-measure is the most frequently used measure for defect classification. Since there are trade-off between precision and recall, comparing different models are not easy as some models have high precision but low recall and vice versa for other models.

VI. DEFECT PREDICTION METRICS

Defect prediction metrics play the most important role to context a statistical prediction model. Most defect prediction metrics can be categorized into two classes: code metrics and process metrics. Code metrics are directly collected existing resource code while process metrics are collected from historical information archived in various software repositories such as version control and issue tracking systems.

A. Code metrics

Code metrics also known as product metrics measure complexity of resource code. Its ground assumption is that complexity resource is more bug-prone. To measure code complexity, researchers proposed various metrics. The size of metrics measure "volume, length, quantity, and overall magnitude of software products". The proxy of size metrics is lines of code (LOC). To our knowledge, Akiyama's model was the earliest study to predict filth using LOC. Afterwards, LOC was used in most defect prediction papers to context a model. Halstead proposed several size metrics based on the number of operators and operands .The proposed metrics are program vocabulary, length, volume, difficult, effort, and time. Most metrics are related to size or quantity. Halstead metrics have been used popularly in many studies. McCabe proposed the cyclomatic metric to portray complexity of software products. Cyclomatic metric is computed by the number of nodes, arcs and connected components in control flow graphs of source code. This metric portray how much control paths are complex. Since McCabe's cyclomatic metric measure the complexity of source code structure, its characteristic is inherently different from size and Halstead metrics that measure volume and quantity of source code. Ohlsson and Alberg adopted McCabe's cyclomatic metric to predict fault-prone modules in telephone switched and other defect prediction studies also used McCabe's cyclomatic metric to build a prediction models.

IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319-5940
International Journal of Advanced Research in Computer and Communication Engineering
Vol. 8, Issue 12, December 2019

B. Process metrics

a) Relative code change churn: Nagappan and ball proposed 8 relative code churn metrics (M1-M8) measuring the amount of code changes. For example, M1 metric is computed by churned LOC (the cumulative number of deleted and added lines between a base version and a new version of a source file) divided by Total LOC. Other metrics (M2-M8) consider various normalized changes such as deleted LOC divided by total LOC, file churned (the number of changed files in a component) divided by file count, and so on. In case study by Nagappan and Ball, the relative churn metrics is proved as a good predictor to explain the defect density of a binary and bug-proneness

b) Change metrics: Change metrics are to measure the extent of changes in the history recorded in version control systems. For example, they can count the number of revisions/bug fix changes/refactoring of a file and the number of authors editing a file. Moser et al. extracted 18 change metrics from the Eclipse repositories to conduct a comparative analysis between code and change metrics. Moser et al.'s change metrics also include added and deleted LOC similar to relative code change churn. However, Moser et al.'s change metrics did not consider any relatedness by the total LOC and the file count but consider average and maximum values of change churn metrics. Moser et al.'s metrics also include maximum and average of change sets (the number of files committed together) and age metrics (age of a file in weeks and weighted age normalized by added LOC). Moser et al. concluded that change metrics are better predictors than code metrics.

c) Change Entropy: Hassan applied Shannon's entropy to capture how changes are complex and proposed history complexity metric (HCM). To validate the HCM, Hassan built statistical linear regression models based on HCM or two change metrics, the number of previous modifications and previous faults on six open-source projects [20]. Their evaluation on six open source projects showed that prediction models built using HCM outperform those using the two change metrics. The idea of adopting the Entropy concept to measure change complexity is novel but comparing models by HCM to those by only two change metrics reveals the weakness of the evaluation of HCM. In addition, evaluation was conducted in the subsystem-level rather than the file-level.

d) Code metric churn, Code Entropy: D'Ambros et al. conducted extensive comparisons study of defect prediction metrics. In their metric comparison, there is no study about code metric churn and code Entropy while code churn and change Entropy metrics have studied as introduced in the previous subsections. Thus, D.Ambros et al. proposed code metric churn (CHU) and code Entropy (HH) metrics.

e) Micro interaction metrics: Lee at al. proposed micro interaction metrics (MIM) extracted from Mylyn that captures developer interactions to Eclipse. The main idea of MIM is from the fact that defects could be introduced by mistakes of developers. For example, more editing time of a certain source code file may cause more bug-proneness. Since Mylyn data contains developer's interactions to Eclipse, Lee et al. extracted 56 metrics from Mylyn data and compared their performance with code and process metrics. In their experiments, MIM outperformed code and process metrics in both classification and regression. However, MIM is highly depended on Mylyn, a plug-in of Eclipse so that MIM might be hard to apply for other development environments that does not support the tool like Mylyn.

VII. DEFECT PREDICTION MODELS

Most defect prediction models are based on machine learning. Depending on what to predict machine learning algorithms are falls into two types namely classification and regression. In the new machine learning techniques, active or semi-supervised learning techniques are applied to build better defect prediction models. Apart from machine learning models, non-statistical model such as Bug Cache has been proposed. Statistical models based on machine learning studies revealed classification and regression models as dominant models. Kim et al. proposed Bug Cache. Studies also investigates case studies. Classification and regression have similar prediction process since they are based on machine learning. The difference between classification and regression models is what to predict. Classification models usually identify bug-proneness while regression models predict the number of bugs. The model used by quality assurance teams depends on the intended purpose of the model users.

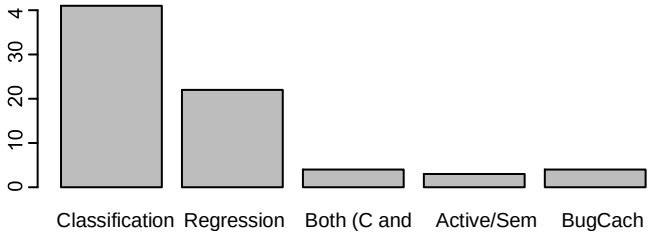

Fig. 3: Use frequency of defect prediction models in the representative defect prediction

Figure 3 shows the use frequency of defect prediction models in the representative defect prediction papers. Classification and regression models are dominant models in machine learning.

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 8, Issue 12, December 2019

VIII. CONCLUSION

Deep study of research articles related to data mining techniques for software defect prediction assures that data mining is an emerging approach for defect prediction. Machine Learning Classifiers have emerged as a way to predict the fault in the software system. Most of these studies performed used different data sets that reflected different software development environment and processes. Various models and techniques studied have association with their process. Data mining techniques can reduce complexity and enhance the performance of the software defect prediction system.

REFERENCES

1. Akiyama. F. An Example of Software System Debugging. In Proceedings of the International Federation of Information Processing Societies Congress, pages 353–359, 1971.
2. Arisholm E., Briand., L. C., and Fuglerud M. Data mining techniques for building fault-proneness models in telecom java software. In Proceedings of the The 18th IEEE International Symposium on Software Reliability, ISSRE '07, pages 215–224, Washington, DC, USA, 2007. IEEE Computer Society
3. Ambrose M. D., Lanza M., and Robbes R. An extensive comparison of bug prediction approaches. In Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on, pages 31 –41, May 2010.
4. Ambrose M. D., Lanza M., and Robbes R. Evaluating defect prediction approaches: A benchmark and an extensive comparison. Empirical Software. Eng., 17(4-5):531–577, Aug. 2012.
5. Bacchelli, M. D'Ambros, and Lanza M. 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.
6. Basili V. R., Briand L. C., and Melo W. L. A validation of object-oriented design metrics as quality indicators. IEEE Trans. Software. Eng., 22:751–761, October 1996.
7. Bird A. Bachmann, Aune E., Duffy J., A. Bernstein, Filkov V., and Devanbu P. Fair and Balanced? Bias in Bug-Fix Datasets. In ESEC/FSE'09, 2009.
8. Bird N. Nagappan., Murphy B., Gall H., and Devanbu P. Don't touch my code examining the effects of ownership on software quality. In Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, ESEC/FSE '11, pages 4–14, New York, NY, USA, 2011. ACM.
9. Chidamber S. R., and Kemmerer C. F. A metrics suite for object oriented design. IEEE Trans. Software. Eng., 20:476–493, June 1994.
10. Conte S. D., Dunsmore H. E., and Shen V. Y. Software Engineering Metrics and Models. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1986.

BIOGRAPHIES

First Author

Y.VANAJA B.Sc., M.Sc., pursuing M.Phil. in Department of Computer Science, Government, Arts College, Coimbatore-641018

Second Author

V.B.Buvaneswari M.Sc., M.Phil., M.E., Assistant Professor in the Department of Computer Science, Government Arts College, Coimbatore-641018.