Fault Prediction Model for Software Using Soft Computing Techniques

Ishrat Un Nisa[1] , Syed Nadeem Ahsan[2]
Dept. of Computer Science, Faculty of Engineering, Science & Technology[1],
Iqra University, Karachi. [1]I.T Department, Sindh High Court, Karachi[2],
ishratbadar@gmail.com, sn_ahsan@yahoo.com

***Abstract*-- Faulty modules of any software can be problematic in terms of accuracy, hence may encounter more costly re-development efforts in later phases. These problems could be addressed by incorporating the ability of accurate prediction of fault prone modules in the development process. Such ability of the software enables developers to reduce the faults in the whole life cycle of software development, at the same time it benefits automation process, and reduces the overall cost and efforts of the software maintenance. In this paper, we propose to design fault prediction model by using a set of code and design metrics; applying various machine learning (ML) classifiers; also used transformation techniques for feature reduction and dealing class imbalance data to improve fault prediction model. The data sets were obtained from publicly available PROMISE repositories. The results of the study revealed that there was no significant impact on the ability to accurately predict the fault-proneness of modules by applying PCA in reducing the dimensions; the results were improved after balancing data by SMOTE, Resample techniques, and by applying PCA with Resample in combination. It has also been seen that Random Forest, Random Tree, Logistic Regression, and Kstar machine learning classifiers have relatively better consistency in prediction accuracy as compared to other techniques.**

Keywords—Data mining; Software metrics; Machine learning; Fault prediction model; Class imbalance; Feature reduction.

I. INTRODUCTION

Defective modules increase development and maintenance cost and also decreases customer satisfaction. Hence it is imperative to predict defective modules as early as possible so that it improves developers' ability to identify the defect prone modules and then focus testing and inspection activities for those modules. Software metrics are very important for measuring the quality of software. It is necessary to predict the quality of software as early as possible, and also collect metrics in early phase [1].

During production of software, its quality assurance takes substantial effort. In order to improve efficiency and productivity of this it is desirable to refer those who need and use it most. Thus we have to identify those chunks of software which are expected to be failed most likely, consequently, require most of our attention [4].

Due to continuous evolution of Software, it is necessary for I.T personnel to update their skills to cope up them with new, enhanced, and complex systems. So they can manage and maintain projects for producing more reliable software by using available resources and within given time [2]. According to Hassan (2009) changing code process in complex way will affect the file with more chances of faults, thus it is a need to avoid complex code change.

Software testing consumes more time and resources; hence it is not desirable to apply same testing effort to all modules of a system, as defects are not uniformly distributed in all parts of a software application. Therefore testers must be able to identify fault prone modules. In this way they become able to prioritize the testing efforts more efficiently. Fault prone modules may examine from their past; if any software component failed in the earlier, it may also be fails in the future [5].

Fault prediction models are useful to reduce the software maintenance time by focusing the development activities on fault-prone code which helps to use the resources in appropriate ways and improve the software quality; hence it is considered as a very important area of research in software engineering, and was the focus of many studies [6]. Therefore, the focus of this study was to propose a ML based fault prediction model that will be used to predict faulty source code modules. Our model is based on: (i) which set of metrics should be used to build a better fault prediction system (ii) how we improve ML prediction model by handling well known techniques of ML such as feature reduction and class imbalance data sets (iii) which classifiers should be used to build a better fault prediction model.

In this study, we used four data sets (AR1, AR5, AR6 & PC3) from publicly available PROMISE Software Engineering Repository[1]. We used 22 metrics, including Halstead, McCabe, and some miscellaneous metrics including defect information. We also reprocess our data for dealing highly correlated and class imbalance issue by using multiple ML techniques such as PCA, Resample and SMOTE in different ways. At the end we tested different ML algorithms (Random Forest, Random Tree, Logistic & Kstar) for designing better fault prediction Model.

The rest of the paper is organized as follows: In Section II we discussed related work; Section III, describes the Research background; Section IV discusses experimental background; Section V consists of results and discussions; finally, in Section VI, we conclude the paper.

[1]http://openscience.us/repo/defect/mccabehalsted/

II. Related Work

Due to the increasing automation activities in human life since last ten years, the demand for quality software is growing day by day. Software with unstable behavior, especially during run time is non tolerable due to its high cost. To ensure high quality, the Software must be reliable; therefore the number of failure must be at a minimum during run time. The primary source of failure is fault in any software program [7]. Thus identification of fault prone entities has a significant impact during software quality assurance. One of the important factors after prediction accuracy is to identify fault prone pieces as early as possible during the development of software[8]. One main reason of faults is updation in source program without analyzing past change patterns. The situation would become severe when developer changes one program file which is logically coupled with other program files and failed to update all other coupled files. Therefore software will not become stable and errors free [9].

According to Zimmermann, et al. (2009) cross project defect prediction modeling plays an important role for projects having insufficient or little data. They further state that it is necessary to quantify, understand, and evaluate the process, data, and domain before the prediction models are built and used. Thus, data and process seemed to be crucial factors since only 3.4% cross project predictions were successful. Jureczko and Madeyski (2010) found that from the defect prediction point of view clusters do exist but only two of those were successfully identified; and these identified clusters cover hardly all software projects. For building models, the methodology seems to be based on predictive performance. The optimal fault prediction model depends on three factors (selection of data with more faulty units, selection of appropriate metrics with applying feature selection and optimization of modeling techniques)[6]. Substantially imbalanced datasets (more non faulty units and a few faulty units) are commonly used in binary fault prediction studies which are present in almost all datasets in NASA MDP. This class imbalancing should not be ignored, it powerfully inspire the training and fitness of the classifier performance. To prevent this many techniques, based on training data can be applied such as sampling or learning algorithm optimization [6]. Shepperd, et al. (2014) states two things about fault prediction. First there is no single prediction technique which may be described as best; secondly prediction results may be changed by using different data sets by applying different preprocessing techniques. Hence there is no single protocol, or algorithm has been suggested for designing a prediction model. Various algorithms are there which can be used by adjusting their parameters to handle imbalanced data; this approach is difficult and time consuming efforts. It is also very difficuilt to select appropriate classifier for the imbalanced data [6]. Jiang, Cukic, et al. (2013) had confirmed that model performance improved by increasing the training data volume, and model designed by using code metrices perform better than design metrics. Therefore both models can be built in different phases of development, whereas model built by using both code and design metrics performed better than their individual models. He concluded it by using 14 data sets from publicly available software engineering data repositories and applied different modeling techniques and statistical tests. Lessman, et al. (2008) considered three possible sources for biasness; he compared classifiers with various branded data sets, considering accuracy indicators which were not appropriate for fault prediction and cross comparison, and statistical testing methods for his findings. Catal and Diri (2009) stated that machine learning based models are better than statistical based model or experts' estimation based models in terms of their features. He also stated that using public datasets is very important becauses of their repeatability, refutability, and verifiability. But when bugs reopened, they not only increases maintenance cost but also reduce quality of software and increases work load, thus developers need to identify re-opened bugs [22].

Data transformation or preprocessing is very important for data mining and machine learning. Jiang, et al. (2008) tested four preprocessing techniques for prediction of fault proneness. He applied ten classification methods on nine datasets taken from NASA Metrics Data Programs (MDP) and investigated that log transformation improves classification performance rarely whereas discretization affects the performance of various algorithms.

III. Research Background

A. Mining Software Repository (MSR)

MSR describes investigation and examination of software repositories that are produced during evolution of software. Software repositories consist of information in source code (concurrent version system), defect information (bug tracking systems), and communication archive (emails) [17].

B. Software Metrics

Software metrics describes all the activities involved in the measurement during whole life cycle in software engineering [18].

C. Machine Learning Issues

In this section, we describe the major issues which are related to the performance of machine learning algorithms.

1) Class Imbalance

In most of the real world applications, the majority of the data related to non-faulty class whereas the researchers are more interested in minority of the data which is related to faulty class. Due to this class imbalance issue, machine learning algorithm cannot perform well on faulty/minority class [19].

2) Feature Selection

Feature Selection is a process for selecting a subset of less correlated attributes/features by eliminating less informative attributes. It not only decreases the size of data but also improves classification accuracy [19].

IV. EXPERIMENTAL BACKGROUND

Description of Data sets and Metrics is as under:

A. Data sets

The present study used four data sets AR1, AR5, AR6 & PC3 from publicly available PROMISE Software Engineering Repository, which was implemented in the C language. This repository stores software metrics along with defect

information of several projects, and the same is available for general users. Each data comprised of various software modules/instances and also include their fault information. All datasets showed bias data (very few faulty instances as compared to large number of non faulty instances), detail of all datasets with their characteristics are listed in Table1.

TABLE 1: DATASETS USED IN THIS RESEARCH

S#	Data	Instances	Faulty (%)	Metrics		Description
				Total	Used	
1	AR1	121	7.44	30	22	Turkish white-goods manufacture (Washing machine)
2	AR5	36	22.22	30	22	Turkish white-goods manufacture (Washing machine)
3	AR6	101	14.85	30	22	Turkish white-goods manufacture (Washing machine)
4	PC3	1458	12.21	38	22	Flight software for earth orbiting satellite

B. Metrics

It is very important to use various metrics, since these measures provide tendency for exploration of the study. All metrics used in this study were already extracted by using Prest tool (Metrics Extraction and Analysis Tool)[2], We have chosen eight Halstead, six McCabe, six miscellaneous, one LOC metrics and one defect information from set of available attributes (30 in AR1,AR5, AR6 and 38 in PC3); all these measures are independent variables (except attribute for defect information) and categorized as code and design level metrics. Table 2 shows 22 different attributes including 21 metrics (independent variables) and one defect information.

TABLE 2: LIST OF METRICS USED IN THIS RESEARCH

S#	Metrics	Type
1	Executable line of code	LOC Based
2	Halstead vocabulary	Halstead
3	Halstead length	Halstead
4	Halstead volume	Halstead
5	Halstead level	Halstead
6	Halstead difficulty	Halstead
7	Halstead effort	Halstead
8	Halstead error	Halstead
9	Halstead time	Halstead
10	Branch count	Miscellaneous
11	Decision count	Miscellaneous
12	Call pairs	Miscellaneous
13	Condition count	Miscellaneous
14	Multiple condition count	Miscellaneous
15	Cyclomatic complexity	McCabe
16	Cyclomatic density	McCabe
17	Decision density	McCabe
18	Design complexity	McCabe
19	Design density	McCabe
20	Normalized cyclomatic complexity	McCabe
21	Formal parameters	Miscellaneous
22	Defects	Defect Info

C. Data Transformation Techniques

We used ML classifiers on all datasets before and after data transformation. We also applied data transformation techniques in combination. In this way we got six groups of datasets as described in Table 3.

Original/Raw Data: The original data set of the repository used without applying any preprocessing method.

PCA: The highly correlated dimensions were reduced by using Principal component analysis.

SMOTE: The original instances were filtered by using Synthetic minority oversampling technique.

Resample Filter: The original data were resampled.

PCA with Resample: Data transformation for feature selection and resolving class imbalance issue in combination were applied.

PCA with SMOTE: Data transformation for feature selection and resolving class imbalance issue in combination were applied.

TABLE 3. DATA REPROCESSING TECHNIQUES USED IN THIS RESEARCH

S#	Formats	Transformation	Techniques
1	Raw Data	-	-
2	PCA	Feature Reduction	Principal Component Analysis
3	Resample	To balance Data	Resample Filter
4	SMOTE	To balance Data	SMOTE Filter
5	PCA with Resample	Feature reduction & Balancing	PCA & Resample
6	PCA with SMOTE	Feature reduction & Balancing	PCA & SMOTE

1) Principal Components Analysis (PCA)

Software metrics, extracted from source code, are mostly highly correlated with each other, since they usually represent measurements or independent variable of related attributes of the software systems. This correlation in metrics often affects prediction capability of model. PCA is a statistical technique to minimize such correlation problem. In PCA, the original metrics are reprocessed and transformed into a smaller set of linear combinations that account for, most of the original data set. It also reduces the quantity of metrics used in building models. These reduced variables are called domain metrics as compared to original independent variables which form the Raw/original metrics. The first principle component shows the largest fraction of the total variance in the original data [15].

[2] http://softlab.boun.edu.tr/?q=resources&i=tools.

2) SMOTE

Synthetic Minority Oversampling Technique is used for reducing class imbalance issue, which resamples given dataset [20].

3) Resample Filter

This technique produces random subsample of the dataset by retaining the same class distribution with or without replacement. We can also mention the percentage of instances in the generated dataset [20].

D. Fault Prediction Algorithms

We build fault prediction models using ML algorithms shown in Table 4. Four classifiers were used with their default parameters such as Random Tree, Random Forest, Logistic regression, & Kstar.

TABLE 4. CLASSIFICATION TECHNIQUES

S#	Classifier	Type	Abbr.
1	Random Tree	Decision Trees	RT
2	Random Forest	Ensemble Decision Trees	RF
3	Kstar	Lazy	KS
4	Logistic Regression	Functions	Lg

E. Building Fault Prediction Models

For building a better fault prediction model (i) set of Halstead and McCabe metrics (from code & design metrics category) have chosen (ii) prediction models were improved by handling well known techniques of ML such as feature reduction and class imbalance as independently and in combination (iii) various classifiers were applied on all groups of data sets (before and after applying transformation techniques) for building better fault prediction model.

In this way the general model building and validation approach adopted in fault prediction modeling with Raw/Actual dataset, applying PCA, SMOTE, Resample, and after combining data transformation techniques (reducing dimensions & class balancing in combination) which summarized in the following steps.

1. Preprocessing Data: All ML classifiers were applied on raw/actual data obtained from repository.

2. Reducing correlated attributes: Highly correlated attributes were reduced by PCA feature reduction technique. All ML classifiers were applied on reduced attributes.

3. Balancing Data: Highly imbalance data were filtered/transformed by using (a) SMOTE & (b) Resample. All ML classifiers were applied again.

4. ML Classifiers: Four ML classifiers were applied on all six groups of four datasets (individually and by a combination of different ML preprocessing techniques).

5. Evaluation of Prediction Models: Models along with the modeling techniques were compared according to their highest values of precision and recall for both faulty and non-faulty classes. Process flow diagram for fault prediction is shown in Figure 1.

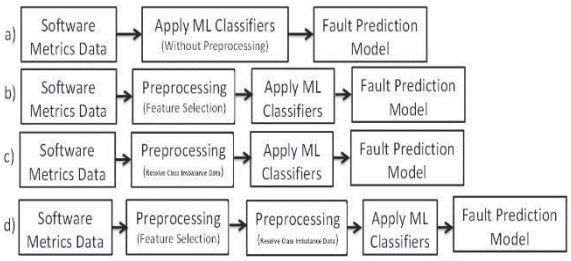

Fig 1. Process Flow Diagram Of Our Approach

F. Precision and Recall

For evaluation of model performance, we used Precision and Recall evaluation measures. Precision, and recall rate are commonly used metrics to assess the binary prediction models. Prediction precision is used to assess the correctness of positive signal predictions; and recall rate is used to assess prediction power for positive signals. In the software engineering field, identifying fault is considered most crucial, therefore recall rate has been considered to be the most important metric [16].

V. RESULTS AND DISCUSSIONS

In this study we focused on both classes (faulty and non-faulty) especially faulty class because correct prediction of faulty class is more crucial as compared to non-faulty class.

Our experiment showed that the results were not significant without preprocessing data but results of all datasets were improved by using class imbalance data transformation techniques Resample and SMOTE, and in combination of feature reduction technique (PCA) with Resample. Whereas all four algorithms provided reliable results but Random Forest and Kstar classify more reliable than random tree and logistic regression especially for faulty classes. The values of Precision and Recall for both faulty and non faulty classes are displayed in Table 5. The results for faulty class of AR5 are also displayed in figure 2. Detail of results is as under:

A. AR1: Random Forest and Random tree performed very well whereas Kstar performed well after resolving class imbalance issue by Resample technique.

Random Tree and Kstar performed very well and Random Forest performed well after resolving both ML issues (feature reduction and class imbalance) by applying PCA with Resample.

AR5: Random forest and Kstar out-performed over other classifiers after resolving class imbalance issue by resample techniques, whereas Random tree performed very well and Logistic Regression performed well after applying Resample technique.

Random Forest, Random tree and Kstar performed very well, and Logistic regression performed well after resolving class imbalance issue by applying SMOTE technique.

All four classifiers (Random forest, Random tree, Logistic regression, and Kstar) performed very well after resolving both

ML issues (Feature selection and class imbalance) by applying PCA with Resample techniques.

Random forest and Logistic regression performed well after resolving both ML issues (Feature selection and class imbalance) by applying PCA with SMOTE techniques.

B. AR6: Random tree, logistic regression, and Kstar classifiers well performed after resolving class imbalance issue by Resample technique.

C. PC3: Kstar performed very well and, Random Forest and Random Tree performed well after resolving class imbalance issue by applying Resample technique.

Random Forest, Random Tree, and Kstar performed well after resolving both ML issues (Feature reduction and class imbalance) by applying PCA with Resample in combination.

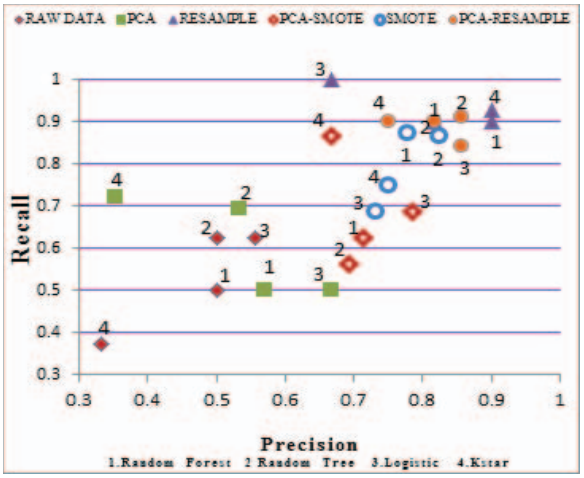

Fig 2. Precision-Recall Values Of Faulty Class Of Ar5

TABLE 5. PRECISION & RECALL VALUES OF FAULTY AND NON-FAULTY CLASSES FOR ALL DATA SETS

Data	ML Tech	Classes	ML Classification results before applying pre-processing techniques		ML Classification results after applying different pre-processing techniques									
					PCA		Resample		PCA with SMOTE		SMOTE		PCA with RESAMPLE	
			Precision (%)	Recall (%)	Precision (%)	Recall (%)	Precision (%)	Recall (%)	Precision (%)	Recall (%)	Precision (%)	Recall (%)	Precision (%)	Recall (%)
AR1	RF	NF	92.50	99.10	93.70	92.90	98.20	99.10	90.60	94.60	90.80	96.40	98.20	97.30
		F	0.00	0.00	20.00	22.20	87.50	77.80	53.80	38.90	63.60	38.90	70.00	77.80
	RT	NF	92.80	92.00	93.60	92.00	99.10	98.20	92.00	92.00	93.30	87.50	98.20	98.20
		F	10.00	11.10	18.20	22.20	80.00	88.90	50.00	50.00	44.00	61.10	77.80	77.80
	Lg	NF	92.60	89.30	93.20	97.30	100.00	94.60	90.20	98.20	91.90	91.10	92.40	97.30
		F	7.70	11.10	25.00	11.10	60.00	100.00	75.00	33.30	47.40	50.00	0.00	0.00
	KS	NF	93.00	95.50	92.90	92.90	98.20	97.30	91.50	96.40	93.00	94.60	98.20	98.20
		F	16.70	11.10	11.10	11.10	70.00	77.80	66.70	44.40	62.50	55.60	77.80	77.80
AR5	RF	NF	85.70	85.70	86.20	89.30	96.20	96.20	80.00	85.70	92.30	85.70	96.00	92.30
		F	50.00	50.00	57.10	50.00	90.00	90.00	71.40	62.50	77.80	87.50	81.80	90.00
	RT	NF	88.50	82.10	87.70	69.60	96.00	92.30	77.40	85.70	88.90	86.60	94.10	91.20
		F	50.00	62.50	53.30	69.60	81.80	90.00	69.20	56.30	82.40	86.60	85.70	91.20
	Lg	NF	88.90	85.70	86.70	92.90	100.00	80.80	83.30	89.30	82.80	85.70	94.10	84.20
		F	55.60	62.50	66.70	50.00	66.70	100.00	78.60	68.80	73.30	68.80	85.70	84.20
	KS	NF	81.50	78.60	80.00	73.40	96.20	93.70	82.80	86.80	85.70	85.70	95.80	88.50
		F	33.30	37.50	35.30	72.30	90.00	92.70	66.70	86.80	75.00	75.00	75.00	90.00
AR6	RF	NF	86.30	95.30	85.90	91.90	93.50	98.90	74.70	86.00	83.30	93.00	93.50	98.90
		F	33.30	13.30	22.20	13.30	87.50	53.80	29.40	16.70	70.00	46.70	87.50	53.80
	RT	NF	87.40	88.40	85.20	87.20	95.60	97.70	78.00	82.60	79.50	81.40	93.30	94.30
		F	28.60	26.70	15.40	13.30	81.80	69.20	40.00	33.30	42.90	40.00	58.30	53.80
	Lg	NF	89.30	87.20	87.50	97.70	95.50	96.60	81.00	94.20	87.50	89.50	92.50	97.70
		F	35.30	40.00	60.00	20.00	75.00	69.20	68.80	36.70	67.90	63.30	75.00	46.20
	KS	NF	87.80	91.90	86.20	94.20	95.70	100.00	74.70	86.00	84.60	89.50	93.50	97.70
		F	36.40	26.70	28.60	13.30	100.00	69.20	29.40	16.70	64.00	53.30	77.80	53.80
PC3	RF	NF	91.20	97.90	90.90	95.90	96.40	98.70	88.00	93.30	89.80	97.30	96.60	98.60
		F	49.10	17.50	31.30	16.30	85.20	66.70	60.00	44.10	81.30	51.60	84.30	68.60
	RT	NF	91.90	90.70	91.90	90.70	96.60	96.40	88.20	87.50	90.40	89.90	96.90	97.00
		F	26.80	30.00	26.80	30.00	68.40	69.20	47.00	48.40	56.70	58.10	72.90	72.40
	Lg	NF	90.90	98.60	90.00	99.30	90.80	99.10	82.90	99.40	85.80	96.20	90.30	99.10
		F	52.50	13.10	37.50	3.80	53.80	9.00	78.00	10.00	64.40	30.00	35.00	4.50
	KS	NF	92.50	92.60	91.10	94.30	97.40	97.40	86.80	93.10	91.60	90.60	96.30	97.50
		F	34.20	33.80	27.30	18.80	76.80	76.30	55.70	38.10	60.60	63.40	74.80	66.70

VI. CONCLUSION

by applying different machine learning techniques and classifiers on various groups of transformed data and comparing different machine learning models, it has been observed that after dealing class imbalance issues by resample and smote techniques especially combining pca with resample, the overall rate of precision and recall become improved. we obtained very good results i.e. ≥ 80% value of precision and recall for both faulty and non-faulty classes in three out of four datasets (ar1, ar5 and pc3) after transformation by pca with resample, whereas in case of ar6, we obtained ≥70% accuracy after transformation by resample. by applying different ml techniques on various dataset, it was observed that random forest, random tree, and kstar provide very good results.

REFERENCES

[1] A, Shanthini, and Chandrasekaran RM. "Applying Machine Learning for Fault Prediction Using Software Metrics." International Journal of Advanced Research in Computer Science and Software Engineering 2, no. 6 (June 2012): 274-284.

[2] Hassan, Ahmed E. "Mining Software Repositories to Assist Developers and Support Managers." 22nd IEEE International Conference on Software Maintenance ICSM'06. . Philadelphia, PA: IEEE, 2006. 339-342.

[3] Hassan, Ahmed E. "Predicting Faults Using the Complexity of Code Changes." Proceedings of the 31st International Conference on Software Engineering. Washington, DC, USA: IEEE Computer Society, May, 2009. 78-88.

[4] Nachiappan, Nagappan, Ball Thomas, and Zeller Andreas. "Mining Metrics to Predict Component Failures." 28th International Conference on Software Engineering (ICSE). New York, NY, USA: ACM, May, 2006. 452-461.

[5] Jureczko, Marian, and Lech Madeyski. "Towards identifying software project clusters with regard to defect prediction." In Proceedings of International Conference on Predictive Models in Software Engineering. New York, NY, USA: ACM, September, 2010.

[6] Hall, Tracy, Sarah Beecham, David Bowes, David Gray, and Steve Counsell. "A Systematic Literature Review on Fault Prediction Performance in Software Engineering." Software Engineering, IEEE Transactions (IEEE Computer Society) 38, no. 6 (Nov-Dec 2012): 1276-1304.

[7] Erturka, Ezgi, and Ebru Akcapinar Sezer. "A comparison of some soft computing methods for software fault prediction." Expert Systems with Applications (Elsevier) 42, no. 4 (March 2015): 1872-1879.

[8] Jiang, Yue, Bojan Cukic, Tim Menzies, and Jie Lin. "Incremental development of fault prediction models." International Journal of Software Engineering and Knowledge Engineering (World Scientic Publishing Company) 23, no. 10 (August 2013): 1399-1425.

[9] Ahsan, Syed Nadeem, and Franz Wotawa. "Fault Prediction Capability of Program File's Logical-Coupling Metrics." Software Measurement, 2011 Joint Conference of the 21st Int'l Workshop on and 6th Int'l Conference on Software Process and Product Measurement (IWSM-MENSURA). Nara: IEEE, 2011. 257-262.

[10] Zimmermann, Thomas, Nachiappan Nagappan, Harald Gall, Emanuel Giger, and Brendan Murphy. "Cross-project Defect Prediction: A Large Scale Experiment on Data vs. Domain vs. Process." 7th Joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The foundations of software engineering. New York, NY, USA: ACM, August 24–28, 2009. 91-100.

[11] Lessman, Stefan , Bart Baesens, Christophe Mues, and Swantje Pietsch. "Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings." IEEE Transactions on Software Engineering, (IEEE) 34, no. 4 (August 2008): 485 - 496.

[12] Shepperd, Martin, David Bowes, and Tracy Hall. "Researcher bias: The use of machine learning in software defect prediction." IEEE Transactions on Software Engineering, (IEEE) 40, no. 6 (June 2014): 603-616.

[13] Catal, Cagatay, and Banu Diri. "A systematic review of software fault prediction studies." Expert Systems with Applications (Elsevier) 36, no. 4 (May 2009): 7346–7354.

[14] Jiang, Yue, Bojan Cukic, and Tim Menzies. "Can Data Transformation Help in the Detection of." In Proceedings of the 2008 workshop on Defects in large software systems. ACM New York, NY, USA, July 2008. 16-20.

[15] Khoshgoftaar , Taghi M., and Naeem Seliya . "Fault Prediction Modeling for Software Quality Estimation: Comparing Commonly Used Techniques." Empirical Software Engineering (Kluwer Academic Publishers) 8, no. 3 (September 2003): 255-283.

[16] Yu, Liguo, and Alok Mishra. "Experience in Predicting Fault-Prone Software Modules Using Complexity Metrics." Quality Technology & Quantitative Management 9, no. 4 (December 2012): 421-433.

[17] Kagdi, Huzefa, Michael L. Collard, and Jonathan I. Maletic. "A survey and taxonomy of approaches for mining software repositories in the context of software evolution." Journal of Software Maintenance and Evolution: Research and Practice (Wiley InterScience) 19, no. 2 (2007): 77-131.

[18] Fenton, Norman E, and Martin Neil. "Software Metrics: Roadmap." Proceedings of the Conference on the Future of Software Engineering. ACM, 2000. 357-370.

[19] Gao, Kehan, and Taghi M. Khoshgoftaar. "Assessments of Feature Selection Techniques with Respect to Data Sampling for Highly Imbalanced Software Measurement Data." International Journal of Reliability, Quality and Safety Engineering (World Scientific Publishing Company) 22, no. 02 (2015): 32.

[20] Witten, Ian H. and Frank, Eibe. Data Mining: Practical Machine Learning Tools and Techniques. San Francisco : Morgan Kaufmann Publishers, 2011.

[22] Studying re-opened bugs in open source software. Shihab, Emad, et al., et al. [ed.] Giuliano Antoniol and Martin Pinzger. 5, s.l. : Springer, 2013, Empirical Software Engineering, Vol. 18, pp. 1005-1042.