IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.Doi Number

Which Process Metrics are Significantly Important to Change of Defects in Evolving Projects: An Empirical Study

LI JIANG[1,2], SHUJUAN JIANG[1,2], (Member, IEEE), LINA GONG[1,2,3], YUE DONG[4], AND QIAO YU[5]

[1]School of Computer Science and Technology, China University of Mining and Technology, Xuzhou, 221116 China
[2]Mine Digitization Engineering Research Center of Ministry of Education, Xuzhou, 221116 China
[3]Department of Information Science and Engineering, Zaozhuang University, Zaozhuang, 277160 China
[4]SunYueqi Honors College, China University of Mining and Technology, Xuzhou, 221116 China
[5]School of Computer Science and Technology, Jiangsu Normal University, Xuzhou, 221116 China

Corresponding author: Shujuan Jiang (e-mail: shjjiang@cumt.edu.cn).

This work was supported in part by the National Natural Science Foundation of China under Grant No. 61673384 and No. 61902161, the Natural Science Foundation of Jiangsu Province under Grant No. K20181353, the Natural Science Foundation of the Jiangsu Higher Education Institutions of China under Grant No. 18KJB520016, and Postgraduate Research & Practice Program of Education & Teaching Reform of CUMT.

ABSTRACT Process metrics can reflect the software development process and the code changes which are the main causes of defects. So, recently, the researches have put more emphasis on process metrics in the field of software defect prediction. For evolving projects, it is more meaningful to study whether the software module introduces or eliminates defects or not, not whether the software module is defective or defect-free. However, no such work is available in the literature focusing on the change of defect state. Discovering the factors that influence the change of defect state in the process of software development can help us to understand the causes of software defects and improve the quality of subsequent software versions. Therefore, this paper presents an extensive empirical study on which process metrics are significantly important to change of defects in evolving projects. Five process metrics of 37 versions in 12 software projects are collected. We not only analyze the class correlation values and the classification performance values among five process metrics, but also perform statistical analysis to verify whether the experimental results are of practical value. The experimental results indicate that Number of Distinct Committers plays a significantly important role in the change of defect state, especially for elimination of defects, and Number of Revisions is the second, whereas Degree of Code Modification is the last. In addition, Average Number of Modified Lines is superior to Number of Modified Lines. Based on the experimental results, some suggestions for software development and software defect prediction are also discussed.

INDEX TERMS Process metrics, software defect prediction, software evolution.

I. INTRODUCTION

During software development and maintenance, requirements changes, bug fixes, and code refactoring can lead to software evolution. Software evolution leads to the increasing scale of software, the increasingly complex relationship among functional modules, and the inevitable defects in software [1]-[3]. Software projects in evolution have a great number of versions of program code. With each new version, there is a possibility to introduce new defects or eliminate previous defects. In fact, the process of software evolution can be regarded as the process of continuously introducing defects and eliminating defects. Defective software can result in serious economic problems, and even endanger human life. Software defect prediction (SDP) can predict defective software modules, and allocate test resources effectively. Recently, software defect prediction has become one of the research hotspots among academic and industrial organizations [3], [4].

Software metrics are indicators or parameters that describe the characteristics of a software product [5] and are input variables for software defect prediction. The performance of defect prediction will be poor if the metrics are not properly selected, causing defect prediction is of little significance. Therefore, this problem, which software metrics should be selected for software defect prediction, has become one of the research hotspots in the field of software defect prediction. Early on, researchers mainly pay attention on code metrics [5]-[12]. Recently, researchers have put more

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2994528, IEEE Access

emphasis on process metrics, such as code churn and people factor, because the code changes generated in software evolution process are the main causes of defects, and process metrics can reflect the software development process and the code changes [13]-[35].

Defect state of a software module includes defective and defect-free, and the change of defect state refers to whether a module introduces or eliminates defects during software evolution, that is, a module introduces defects or eliminates defects from the completion of the previous version to the completion of the current version. For evolving projects, the new class may introduce defects or be defect-free, and the existing class may change from defect-free to defective, or from defective to defect-free, and may still be defective or defect-free. For example, a module is defect-free in the previous version, but is defective in the current version, we can consider it as an instance of introduction of defects. On the contrary, a module is defective in the previous version, and is defect-free in the current version, we can consider it as an instance of elimination of defects. Of course, there are other states as well, for example, a module is defect-free in the previous version, and still is defect-free in the current version, and a module is defective in the previous version, and still is defective in the current version. However, we are more concerned with "elimination of defects" and "introduction of defects", so we put all the others into one category "others". Therefore, the change of defect state of a class can be divided into three categories: elimination of defects, introduction of defects, and others. Eliminating defects indicates that software is evolving in a good direction, and improper evolution leads to introducing defects, which we do not want to observe when software evolves.

Previous researches on software defect prediction tend to start from the perspective of whether there is a defect or not, that is, the classes of classification are defective and defect-free. However, for evolving projects, it is more meaningful to study whether the software module introduces or eliminates defects or not, that is, the change of defect state, because it is more beneficial for us to find the problems in the process of software development for evolving software. And no such work is available in the literature focusing on the change of defect state. Therefore, in this paper, we focus on the change of defect state of software modules, that is, the class of experiment datasets is the change of defect state. Discovering the factors that influence the change of defect state in the process of software development can help us to understand the causes of software defects and improve the quality of subsequent software development.

This paper presents an empirical study on which process metrics are significantly important to change of defects in evolving projects. In general, the contributions of this paper are summarized as follows:

i) The existing researches of software defect prediction for evolving projects mainly focused on whether the module is defective or not. This paper focuses on whether the software module introduces or eliminates defects or not. This paper presents an empirical study on which process metrics are significantly important to change of defects in evolving projects. To the extent of our knowledge, no earlier work is available that has explored the factors that influence the change of defect state in the process of software development for evolving projects.

ii) We study on which process metrics are significantly important to change of defects in evolving projects from two aspects. First, we compare the class correlation values among five process metrics by using six class correlation measurement methods, including Pearson Correlation Coefficient, Chi-Square, ReliefF, Information Gain, Gain Ratio, and Symmetric Uncertainty. Second, we compare the classification performance values among five process metrics in terms of four evaluation measures, including Recall, F-Measure, AUC, and MCC, by using five classification algorithms, including Naive Bayes, K-nearest Neighbor, Logical Regression, Multilayer Perceptron, and Support Vector Machine.

iii) Additionally, we conduct the empirical study for the project datasets extracted by Madeyski et al. [19], including 18 releases of seven open source and 19 releases of five industrial software projects. Very limited works are available earlier where these projects have been used for the comparison of process metrics in defect prediction of evolving projects.

iv) To do the evaluation of the class correlation and classification performance values among five process metrics, we perform an experimental analysis by using different class correlation measurement methods and classification algorithms. We not only analyze the class correlation values and the class performance values among five process metrics, but also perform statistical analysis of Wilcoxon matched-pair signed-rank test and Cohen's d to verify whether the experimental results are statistically significant and calculate the effect size. The experimental results indicate that Number of Distinct Committers (NDC) plays a significantly important role in the change of defect state including introduction of defects and elimination of defects, and Number of Revisions (NR) is the second, whereas Degree of Code Modification (DCM) is the last. In addition, Average Number of Modified Lines (ANML) is superior to Number of Modified Lines (NML). Based on the experimental results, some suggestions for software development and software defect prediction are also discussed.

The organizational structure of this paper is as follows. The related works of process metrics and evolutionary defect prediction are discussed in section II. Section III describes the case study in detail. The experimental results and analysis are presented in section IV. The threats to our study are discussed in section V. Lastly, the summary and future works are presented in section VI.

II. RELATED WORKS

In this section, we mainly introduce the research status of process metrics and defect prediction of evolving projects.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2994528, IEEE Access

A. PROCESS METRICS

We can extract software metrics from databases generated during software development, such as code repositories, defect repositories, and version control systems. Based on the common characteristics of software defective modules, the researchers proposed a variety of software metrics related to software scale, software complexity, and human psychology, which can be divided into code metrics and process metrics.

Code metrics were wildly used for defect prediction early. This kind of researches thought that code size and code complexity which were easy to extract is strongly related to defects in software, such as the LOC metric representing code size [5], the McCabe metrics describing software code structure complexity [6], the Halstead metrics representing code complexity defined by the number of operands and operators [7], the CK metrics of object-oriented programs [8], and the metrics of abstract syntax tree [9]. Many scholars have criticized code metrics. Olague *et al.* [10] used logistic regression to implement cross-version defect prediction based on a variety of object-oriented metrics, and found that the prediction performance of object-oriented metrics is not ideal. Shepperd *et al.* [11] and Radjenović *et al.* [13] pointed out that using LOC metrics and complexity metrics at the same time may affect the performance of defect prediction, because many complexity metrics are strongly correlated with LOC. Rahman *et al.* [14] pointed out that code metrics have hysteresis, that is, code metrics may not be changed much after fixing bugs. Consequently, simply using code metrics is not suitable for defect prediction in evolving projects.

Different from the information reflected by code metrics, process metrics directly reflect the software development process and software evolution track. The code changes generated in the evolution process are the main causes of the defects in evolving projects. Recently, researchers have put more emphasis on process metrics. A considerable process metrics are proposed, mainly including: i) metrics based on code change history, such as number of modified lines [13], [19], [20], [23], [28]-[33], and code relative change metrics [13], [20], [27], [30], ii) metrics based on developer information, such as number of distinct committers [15], [19], [20], [23]-[26], [30], [32], [34], experience of developers [16], [34], commit activities of developers [34], project team organizational structure [17], code ownership [34], and organizational dispersion degree [18], iii) development process related metrics, such as number of revisions [13]-[15], [19], [20], [22], [23], [25], [30], number of defects repaired [30], number of refactorings [20], [30], code change complexity [21], [32], and number of historical defects [19], [23]. The most widely used, classical, and defect-related process metrics are Number of Revisions (NR), Number of Distinct Committers (NDC), Number of Modified Lines (NML), and code relative change metrics. The research status of these process metrics are as follows:

- NR is a widely used process metric in software defect prediction. Schröter *et al.* [15] found that NR was more relevant to the number of defects. Graves *et al.* [22] showed that NR was a better predictor of defect, at least better than LOC. Illes-Seifert *et al.* [25] compared the correlation between process metrics and the number of defects. The experimental results showed that NR is strongly correlated with the number of defects, and NR has better defect prediction performance.
- NDC is a controversial process metric. Some researchers thought the introduction of NDC had no effect on improving the performance of defect prediction model. Weyuker *et al.* [24] compared the performance of the prediction models with and without NDC, and found that NDC could not significantly improve the prediction performance. There are also some researchers who claimed NDC can improve the defect prediction performance. Illes-Seifert *et al.* [25] found NDC was highly related to the number of defects, and could obtain great defect prediction performance in terms of predicting the number of defects. Matsumoto *et al.* [26] analyzed the correlation between a variety of developer metrics and the number of defects, and evaluated the influence of developer metrics on the defect prediction performance with the number of defects as prediction target. The experimental results showed that the introduction of developer related information could improve the defect prediction performance. Kini *et al.* [34] extracted periodic developer experience metrics at file level and commit level, and investigated the explanatory effect of these metrics on defects. The experimental results showed that periodic developer experience metrics extracted at file level were good merits for defect prediction.
- NML is also a widely used process metrics in software defect prediction. Previous studies have shown that NML was strongly related to defects. Nagappan *et al.* [28] pointed out that NML had a good defect density prediction performance. Shin *et al.* [29] also showed that NML had better defect tendency prediction performance. Liu *et al.* [31] proposed an NML based unsupervised defect prediction model (CCUM) in effort-aware JIT defect prediction, and evaluated the prediction performance of CCUM under cross validation, time-wise cross validation, and cross-project validation. The experimental results showed that CCUM performed better than all the prior supervised and unsupervised models. Miletić *et al.* [33] built standard prediction models with and without cross-version code churn. The prediction models were trained on earlier releases and tested on the following ones, and the experimental results showed that the prediction model performed better when cross-version code churn was included.
- The concept of relative was first proposed by Munson *et al.* [35]. Code relative change metrics here refer to the degree of code change between two adjacent versions, usually represented by the ratio of NML to other software metric. Nagappan *et al.* [27] conducted a study on the defect prediction performance of eight code relative change metrics. The experimental results showed that

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2994528, IEEE Access

compared with code absolute change metrics, the code relative change metrics could obtain better defect density prediction performance and defect tendency prediction performance.

Some researchers compared the defect prediction performance of process metrics with the defect prediction performance of code metrics, and found that the defect prediction performance of process metrics or their combination was better than that of code metrics. Radjenović *et al.* [13] compared their defect prediction performance, and the experimental results showed that the prediction performance of process metrics was better. Moreover, the better process metrics were code relative change metrics, NR, and NML. Moser *et al.* [20] compared their defect prediction performance, using 17 process metrics, including NR, NDC, and NML. The experimental results showed that the prediction performance of these process metrics was significantly better than that of code metrics, and the prediction performance of these process metrics was similar to that of the combination of code metrics and process metrics. Graves *et al.* [22] found that for defect density prediction, the prediction performance of metrics based on code change history was better than that of code metrics. Madeyski *et al.* [19] conducted an empirical study on identifying which process metrics could significantly improve the performance of defect prediction models, using NR, NDC, NML, and number of defects in previous version (NDPV) as process metrics. First, they analyzed the correlation between each process metric and the number of defects. Then, they compared the performance of the models which used only code metrics with that of the models which used code metrics as well as one of the process metrics. The experimental results showed that the introduction of process metrics could significantly improve the performance of defect prediction, especially NDC. Based on this, Stanić *et al.* [23] conducted comparative experiments on software metrics. The experimental results showed that the prediction performance of process metrics was better than that of code metrics, and there was no significant difference in the prediction performance of different combinations of code metrics and process metrics. Choudhary *et al.* [30] proposed new change metrics and extracted change metrics from the GIT repositories. The change metrics they studied included NR, NDC, NML, ANML, average number of commits made by each developer, average lines of code worked by each developer, and so on. Machine learning algorithms were applied in change metrics and code metrics to build fault prediction models. The experimental results demonstrated that the use of change metrics in conjunction with code metrics provided better performance than the models that had individual metrics set, and the change metrics had a positive impact on the prediction performance.

B. DEFECT PREDICTION OF EVOLVING PROJECTS

Defect prediction of evolving projects is also called cross-version defect prediction. Defect prediction model is built on the previous versions, and is used to predict the defects in the current version. The classical defect prediction models include Naïve Bayes (NB) [36], K-Nearest Neighbor (KNN) [37], Logistic Regression (LR) [38], Support Vector Machine (SVM) [39], and Multilayer Perceptron (MLP) [40]. The main researches on defect prediction of evolving projects can be divided into the following two types.

Some researches mainly focus on using existing and newly proposed machine learning algorithms to build cross-version defect prediction models, and evaluating defect prediction performance under cross-version scenario. Yang *et al.* [41] compared the prediction performance of RR and LAR with other classical algorithms under cross-version scenario in terms of predicting the number of defects. Shukla *et al.* [42] regarded cross-version defect prediction as a multi-objective optimization problem for the first time, considering both the prediction performance and the cost. The experimental results showed that the multi-objective optimization algorithm had a broad application prospect in defect prediction of evolving projects. Martino *et al.* [43] used genetic algorithm to search the optimal parameter configuration for SVM to improve the prediction performance. The experimental results showed that the performance of proposed algorithm was better than the comparison algorithms in cross validation scenario and cross-version validation scenario. Liu *et al.* [44] proposed a recursive neural network prediction model with the sequence of all metrics in the continuous history version as the input. The experimental results showed that in most cases, the proposed HVSM-based RNN model had a significantly better effort-aware ranking effectiveness than baseline models. Rathore *et al.* [45] presented an approach that dynamically selected the best learning techniques to predict the number of software faults. The approach partitioned the validation dataset into different module subset and determined the best learning technique for each subset. For an unseen testing module, the approach determined the subset from the validation dataset that had modules similar to the given testing module, and the best learning technique for the determined subset was the best learning techniques for the testing module. They built and evaluated the presented approach for intra-release prediction and inter-releases prediction, demonstrating the effectiveness of the approach for cross-version software defect prediction.

Other researchers proposed algorithms to solve the problem of data distribution inconsistency between the source dataset and the target dataset in cross-version defect prediction. Active learning was introduced into cross-version software defect prediction to solve the problem of inconsistent data distribution. Lu *et al.* [46] proposed an approach which took uncertain information as the strategy to select special instances of the current version iteratively, and then determine their classes manually and merged them into the training set. Xu *et al.* [47] proposed an active learning method based on uncertainty information and information density to select special instances from the current version.

From the research status of process metrics and defect prediction of evolving projects, we can observe that no such

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2994528, IEEE Access

work is available in the literature focusing on the factors that influence the change of defect state in evolving projects. This paper presents an empirical study on which process metrics are significantly important to change of defects in evolving projects. The process metrics used in this paper are NR, NDC, NML, DCM, and ANML which are widely used, classical, and defect-related.

III. CASE STUDY

First, we describe the code metrics and process metrics used in this paper. Then, we provide the experimental datasets. Next, we give the data preprocessing operation. Finally, we report the experimental design.

A. SOFTWARE METRICS

Software metrics can be divided into code metrics and process metrics. The former describes the scale and complexity of software source code, and the latter describes the complexity of software development process [14], [15].

Code metrics used in this paper include: i) code size metric (LOC), ii) McCabe cycle complexity metric, iii) object-oriented metric [12]. These metrics can be extracted from the source code by Ckjm[1] tool.

As NR, NDC, NML, DCM, and ANML are the most classical and widely used process metrics, and they are strongly related to defects, we select them as the experimental objects. The first three are the same as those in literature [19], and can be extracted from SVN and CVS by using BugInfo [2] tools. The last two were proposed in literature [27] and [20] respectively, and can be calculated by the ratio of NML to other software metric. The following gives a detailed description of these five process metrics.

- Number of Revisions (NR) is a metric related to the development process, which refers to the total number of versions submitted by software developers to the version control system from the completion of the previous version to the completion of the current version.
- Number of Distinct Committers (NDC) is a metric based on developer information, which refers to the total number of developers participating in the development from the completion of the previous version to the completion of the current version.
- Number of Modified Lines (NML) is a metric based on code change history, which refers to the total number of added, deleted, and modified code lines submitted from the completion of the previous version to the completion of the current version.
- Degree of Code Modification (DCM), obtained by dividing the number of modified lines by the total number of lines, i.e. DCM = NML / LOC, is one of the code relative change metrics. DCM represents the degree of code modification, that is, the average times each line of code has been modified.
- Average Number of Modified Lines (ANML), obtained by dividing the number of modified lines by the number of revisions, i.e. ANML = NML / NR, is one of the code relative change metrics. ANML represents the degree of code modification, that is, the number of modified lines involved in each submission.

B. EXPERIMENTAL DATASETS

NR, NDC, NML, and 20 code metrics of this study are downloaded from the database[3] created by Madeyski *et al.* [19], whereas DCM and ANML are calculated based on the existing data. We determine the classes of each instance based on whether the previous version has defects or not and whether the current version has defects or not. And the classes of these datasets include “introduction of defects”, “elimination of defects”, and “others”. So, each instance of these datasets is a software module in the current version, consisting of 20 code metrics, five process metrics, and a class, that is, the change of defect state from the previous version to the current version.

These project versions in the database meeting following three conditions are removed from the dataset created by Madeyski *et al.* [19]. i) The first version of each project does not have evolution history and five process metrics, so we do not list those project versions, such as ant-1.3 and camel-1.0. ii) It is unable to collect all five process metrics for some projects, including poi, pbeans, ivy, log4j, velocity, prop-1-192, prop-2-225, prop-3-285, prop-4-347, prop-5-185, and prop-6. So, we also do not use them as experimental objects. iii) Some project versions with very high class imbalance rate have very few instances of introduction of defects or elimination of defects, including camel-1.2 which has only two instances of elimination of defects, 205 instances of introduction of defects, and 558 instances of others, xalan-2.7, and xerces-1.4.4. Such a high class imbalance rate and a small number of minority class will affect the experimental results, so we do not use these project versions as experimental datasets. The remaining projects including seven open source software projects with 18 versions and five commercial projects with 19 versions are the experimental subjects. These software projects are all Java projects and from different application fields.

Table I lists the detailed information of these experimental datasets. The first to three columns are the project name, version number, the number of all classes in the version respectively. The fourth and sixth columns are the number of the instances from defect-free to defective (introduction of defects) and the number of the instances from defective to defect-free (elimination of defects) respectively. The fifth and seventh columns are the proportion of the instances from defect-free to defective and the proportion of the instances from defective to defect-free respectively.

From Table I, we can observe that the process of software evolution is the process of continuously introducing defects and eliminating defects. In some versions, most of classes eliminate defects, and a few of classes introduce defects, such as prop-1-44 and prop-2-265. More classes eliminate

[1] https://www.spinellis.gr/sw/ckjm/

[2] https://kenai.com/projects/buginfo - not available now

[3] http://madeyski.e-informatyka.pl/tools/software-defect-prediction/

defects and fewer classes introduce defects indicates that the software is evolving towards a better direction. In other versions, most of classes introduce defects, and a few of classes eliminate defects, such as ant-1.4 and ant-1.6. This shows that these projects evolved improperly. These 37 datasets have the same 20 code metrics and five process metrics, and they have different proportions of introduction of defects and elimination of defects. So, these datasets can be used as the experimental datasets to evaluate the influence of these five process metrics on the change of defect state in evolving projects.

TABLE I. The experimental datasets

Project	Version	Number of all classes	Number of defect-free → defective (introduction of defects)	Proportion of defect-free → defective	Number of defective → defect-free (elimination of defects)	Proportion of defective → defect-free
ant	1.4	265	34	12.8%	14	5.3%
	1.5	401	25	6.2%	32	8.0%
	1.6	523	75	14.3%	15	2.9%
	1.7	1066	104	9.8%	29	2.7%
camel	1.4	1122	48	4.3%	113	10.1%
	1.6	1252	106	8.5%	62	5.0%
jedit	4.0	606	28	4.6%	43	7.1%
	4.1	644	30	4.7%	24	3.7%
	4.2	805	19	2.4%	49	6.1%
	4.3	1132	8	0.7%	44	3.9%
lucene	2.2	381	79	17.4%	25	6.6%
	2.4	536	99	18.5%	39	7.3%
synapse	1.1	230	51	22.2%	7	3.0%
	1.2	269	56	20.8%	28	10.4%
xalan	2.5	945	316	33.4%	30	3.2%
	2.6	1170	190	16.2%	164	14.0%
xerces	1.2	515	39	7.6%	35	6.8%
	1.3	545	52	9.5%	51	9.4%
prop-1	9	4455	101	2.3%	324	7.3%
	44	4081	10	0.2%	891	21.8%
	92	3670	1152	31.4%	83	2.3%
	128	3619	145	4.0%	221	6.1%
	164	3541	256	7.2%	21	0.6%
prop-2	236	2403	62	2.6%	133	5.5%
	245	2023	88	4.3%	57	2.8%
	256	2025	588	29.3%	66	3.3%
	265	2372	160	6.7%	543	22.9%
prop-3	292	2330	160	6.9%	125	5.4%
	305	2388	55	2.3%	171	7.2%
	318	2440	334	13.7%	57	2.3%
prop-4	355	2802	854	30.5%	76	2.7%
	362	2865	122	14.8%	824	28.8%
prop-5	4	3514	175	5.0%	247	7.0%
	40	3815	54	1.4%	515	13.5%
	85	3509	707	20.1%	193	5.5%
	121	3445	279	8.1%	213	6.2%
	157	2863	258	9.0%	153	5.3%

C. DATA PREPROCESSING

Proper data preprocessing can improve the quality of the dataset, and is beneficial to experimental research, so we conduct data preprocessing operations, including data normalization processing and class imbalance processing.

Different metrics have different ranges, which may affect the relationship between each metric and the change of defect state. To alleviate the negative impact of different ranges on the evaluation results, we use Maximum-Minimum method [48] to normalize the values of all metric to [0, 1]. The equation of data normalization processing is as follows:

$$M_{ij}^{'} = \frac{M_{ij} - Min(M_i)}{Max(M_i) - Min(M_i)} \quad (1)$$

Where $M_{ij}^{'}$ represents the value of the i^{th} metric of the j^{th} instance after normalization, M_{ij} represents the value of the i^{th} metric of the j^{th} instance before normalization, $Max(M_i)$ represents the maximum value of the i^{th} metric of all instances, $Min(M_i)$ represents the minimum value of the i^{th} metric of all instances.

The experimental datasets suffer from varying degrees of class imbalance problem, and the class imbalance problem can influence the experimental results. We use SMOTE (Synthetic minority over-sampling technique) [49] to handle the class imbalance problem of these datasets. SMOTE generates synthetic numeric values for the minority classes only, to balance the number of the instances of the minority classes with that of the majority class. Generating synthetic values refers to generating new values from the existing instances in the dataset. For example, for each minority class instance X, SMOTE selects an instance X_k from the nearest neighbors of X randomly, and then selects a point randomly on the line between X and X_k as the newly synthetic instance of minority class. More description of SMOTE can be referred from the work presented by Chawla *et al*. [49]. The synthetic value $X^{'}$ is given by (2).

$$X^{'} = X + (X_k - X) \times \delta \quad (2)$$

Where δ is a random number between 0 and 1.

D. EXPERIMENTAL DESIGN

The aim of this paper is to compare the importance of a single process metric to the change of defect state, including introduction of defects and elimination of defects, in evolving projects. Figure 1 provides an overview of our empirical study.

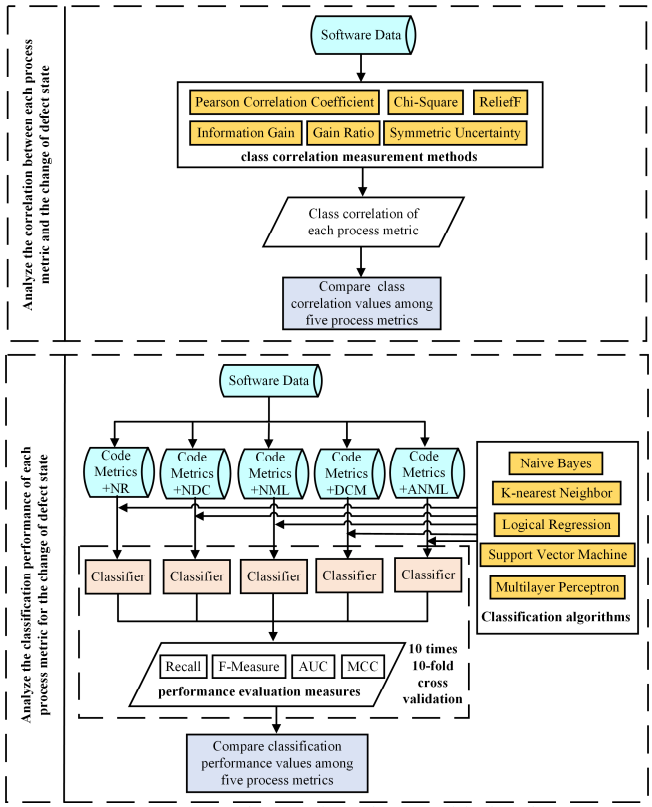


FIGURE 1. The overview of our empirical study

According to the overview of our empirical study, to study on which process metrics are more important to the change of defect state, we need to analyze the correlation between each process metric and the change of defect state and analyze the classification performance of each process metric for the change of defect state. The following research questions are addressed in this study.

RQ1: How is the correlation between each process metric and introduction or elimination of software defects?

RQ2: What is the ability of each process metric to classify introduction of defects and elimination of defects?

For RQ1, there are many methods to calculate the class correlation, which can be divided into three categories: i) methods based on statistical theory, such as Pearson Correlation Coefficient and Chi-Square, ii) methods based on instances, such as Relief and ReliefF, iii) methods based on information entropy theory, such as Information Gain, Gain Ratio, and Symmetric Uncertainty. We use six classical methods of class correlation measurement, including Pearson Correlation Coefficient, Chi-Square, ReliefF, Information Gain, Gain Ratio, and Symmetric Uncertainty, which cover all the above three categories, to conduct the class correlation analysis. This experiment is implemented by Weka[4], a specialized tool for machine learning and data mining, to ensure these class correlation measurement methods are accurate.

Pearson Correlation Coefficient is a method to evaluate the importance of a metric to the classification by measuring the linear correlation between each metric and the classes. Pearson Correlation Coefficient of variable X and variable Y is given by (3).

$$Pearson(X,Y)=\frac{\mathrm{cov}(X,Y)}{\sigma_X\sigma_Y}=\frac{\mathrm{E}((X-\mu_X)(Y-\mu_Y))}{\sigma_X\sigma_Y}=\frac{\sum XY-\frac{\sum X\sum Y}{N}}{\sqrt{\sum(X^2)-\frac{(\sum X)^2}{N}}\sqrt{\sum(Y^2)-\frac{(\sum Y)^2}{N}}} \quad (3)$$

Chi-Square is a kind of nonparametric statistical value used to verify whether a metric is related to the class distribution. The null hypothesis is that they are not related. Then, the possibility of null hypothesis is measured by calculating the distance between the observed value and the expected value when null hypothesis is established. The greater the distance is, the less possible the null hypothesis is, and the more likely it is that the metric distribution is related to the class distribution. Chi-Square is given by (4).

$$\chi^2=\sum_{i=1}^{r}\sum_{j=1}^{n_c}\frac{(O_{i,j}-E_{i,j})^2}{E_{i,j}} \quad (4)$$

Where r respects different values of a metric, n_c is the number of classes, O_{ij} and E_{ij} respect the observed number of instances and the expected number of instances whose metric value is i in class j.

Unlike the first two methods, ReliefF does not directly calculate the correlation between the metric and the classes, but gives each metric an importance coefficient which measures the ability of the metric to distinguish the instances of different classes, and then updates the coefficient iteratively. The coefficient is given by (5).

$$W(A)=W(A)-\sum_{j=1}^{k}diff(A,S,H_j)/(mk)+\sum_{C\in class(S)}\left[\frac{p(C)}{1-p(class(S))}\sum_{j=1}^{k}diff(A,S,M_j(C))\right]/(mk) \quad (5)$$

Where $diff(A,R_1,R_2)$ is the distance between instance R_1 and R_2 on the metric A, H_j is the instance with the same class as R, $M_j(C)$ is the instance with the different class from R, m is the times R is randomly selected, and k is the number of the nearest neighbor instances of R selected each time.

Information Gain measures the amount of information brought for classification by the metric about the classes, which is calculated by the information entropy of original dataset S minus the information entropy of datasets divided by metric A. The value of Information Gain is given by (6).

$$IG(S\mid A)=H(S)-H(S\mid A) \quad (6)$$

Where H is the information entropy.

Information Gain Ratio introduces split information on the basis of Information Gain, which offsets the impact of the number of the metric values on the amount of Information brought for classification by the metric. The value of

[4] https://www.cs.waikato.ac.nz/ml/weka/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2994528, IEEE Access

Information Gain Ratio is given by (7).

$$GR(A) = \frac{IG(S \mid A)}{SplitE(A)} \tag{7}$$

Symmetric Uncertainty is a nonlinear correlation measurement method. Symmetric Uncertainty of the metric A and the classes is given by (8).

$$SU(S \mid A) = 2 \times \frac{IG(S \mid A)}{H(S) + H(A)} \tag{8}$$

The process of class correlation analysis of process metrics is shown in Procedure 1. First, two data preprocessing operations are conducted on 37 datasets, including data normalization processing and class imbalance processing (lines 2 to 3). And then, the correlation between each process metric and the change of defect state is evaluated by each class correlation measurement method (line 5). Last, the ranking of process metrics is obtained according to their class correlation (line 6).

Procedure 1: Class Correlation Analysis of Process Metrics

Input: *processMetric* ∈ {NR, NDC, NML, DCM, ANML},
DATA ∈ {D_1, D_2, …, D_{37}},
Selector ∈ {Pearson Correlation Coefficient, Chi-Square, ReliefF, Information Gain, Gain Ratio, Symmetric Uncertainty}
Output: Class correlation values of each process metric and *Ranker*.

1 **Begin**
2 **for each** *DATA* **do**
3 *DATA'* ← preprocess *DATA*; /* normalization, class imbalance processing */
4 **for each** *Selector* **do**
5 *correlation* ← *Selector*(*DATA'*); /* calculate class correlation value of each *processMetric* */
6 *Ranker* ← sort process metrics according to class correlation;
7 **end for**
8 **end for**
9 **End**

For RQ2, we need to construct classification models. Lessmann *et al.* [4] showed that most of the classification algorithms had similar performance and had no significant difference. Therefore, we select five classical and effective classification algorithms: Naive Bayes (NB) [36], K-nearest Neighbor (KNN) [37], Logical Regression (LR) [38], Support Vector Machine (SVM) [39], and Multilayer Perceptron (MLP) [40] to construct the classification models of introduction of defects and elimination of defects, and verify the consistency of five classification algorithms. Similarly, this experiment is implemented by Weka. For KNN, K is set to 5, and for other algorithms, we use the default parameters of Weka.

In this paper, we use the combination of all code metrics and each process metric to build the classification model. 10 times 10-fold cross validation is used. First, divide a dataset into ten equal parts. Then, select nine of them as training set, the remaining one as testing set, and repeat 10 times to ensure that each part is tested. Next, the average value of 10 times is taken as final performance. Finally, repeat the above three processes 10 times to alleviate the effect of randomness. We select Recall, F-Measure, AUC, and MCC as performance evaluation measures to compare the performance of these classification models. Because we focus on introduction of defects and elimination of defects, eight evaluation measures are used in total, including Recall-Introducedefects, Recall-Removedefects, F-Measure-Introducedefects, F-Measure-Removedefects, AUC-Introducedefects, AUC-Removedefects, MCC-Introducedefects, and MCC-Removedefects. These performance evaluation measures can be calculated by the confusion matrix, as shown in Table II.

TABLE II. The confusion matrix

	classification as class k	classification as not class k
actually class k	TP_k	FN_k
actually not class k	FP_k	TN_k

Recall refers to the ratio of the number of instances correctly predicted as class k to the total number of instances with class k, that is, true positive rate. It is defined as

$$Recall_k = \frac{TP_k}{TP_k + FN_k} \tag{9}$$

False positive rate (*pf*) is the ratio of the number of instances incorrectly predicted as class k to the total number of instances which are not of class k, and is shown as follows:

$$pf_k = \frac{FP_k}{FP_k + TN_k} \tag{10}$$

Precision is the ratio of the number of instances correctly predicted as class k to the total number of instances predicted as class k, and is shown as follows:

$$Precision_k = \frac{TP_k}{TP_k + FP_k} \tag{11}$$

F-Measure is the harmonic average of Recall and Precision, and is shown as follows:

$$F\text{-}Measure_k = \frac{2 \times Recall_k \times Precision_k}{Recall_k + Precision_k} \tag{12}$$

Area Under the Curve (AUC) is the area under Receiver Operating Characteristic curve (ROC). The curve describes the relationship between true positive rate and false positive rate. The abscissa represents false positive rate, and the ordinate represents true positive rate. Each point on the curve corresponds to a classification threshold. AUC is not affected by class imbalance as well as is independent from the prediction threshold.

Matthews Correlation Coefficient (MCC) represents the correlation coefficient between actual classification and prediction classification. MCC is calculated from four values in the confusion matrix. It is defined as

$$MCC_k = \frac{TP_k \times TN_k - FP_k \times FN_k}{\sqrt{(TP_k + FP_k)(TP_k + FN_k)(TN_k + FP_k)(TN_k + FN_k)}} \tag{13}$$

The values of Recall, F-Measure, and AUC range from 0 to 1, and the MCC ranges from -1 to 1. The higher the value is, the better the performance of classification model is.

The process of classification performance analysis of process metrics is shown in Procedure 2. First, two data preprocessing operations are conducted on 37 datasets, including data normalization processing and class imbalance processing (lines 2 to 3). And then, the process metric to be evaluated is remained, and other process metrics are removed, that is, the combination of all code metrics and each process metric are used to build each classification model (line 5). Last, we conduct 10 times 10-fold cross

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2994528, IEEE Access

validation to build and evaluate the performance of each classification model (lines 6 to 21). First, the order of instances is upset, and the dataset is divided into ten equal parts (lines 7 to 8). Secondly, we use nine parts as training data, and the left one as testing data in turn (lines 9 to 11). Thirdly, we use five classification algorithms to train each classifier on training set respectively (lines 12 to 13). Fourthly, we use eight performance evaluation measures to evaluate classification performance of each classifier (lines 14). Last, we repeat the above processes ten times, and take the average value of ten times as the classification performance of each process metric (lines 18 to 20).

Procedure 2: Classification Performance Analysis of Process Metrics

Input: *processMetric* ∈ {NR, NDC, NML, DCM, ANML},
DATA ∈ {D_1, D_2, …, D_{37}},
Learner ∈ {NB, KNN, LR, MLP, SVM}
Output: *Measure* ∈ {Recall-Introducedefects, Recall-Removedefects, F-Measure-Introducedefects, F-Measure-Removedefects, AUC-Introducedefects, AUC-Removedefects, MCC-Introducedefects, MCC-Removedefects}

```
1  Begin
2  for each DATA do
3    DATA' ← preprocess DATA;   /* normalization, class imbalance
                                   processing */
4    for each processMetric do
5      DATA'' ← reserve all code metrics and metric, and delete
                other process metrics of DATA';
6      for each times ∈ [1, 10] do   /*10 times 10-fold cross
                                       validation*/
7        DATA''' ← randomize the order of instances for DATA'';
8        binData ← generate 10 bins from DATA''';
9        for each fold ∈ [1, 10] do
10         testingData ← binData[fold];
11         trainingData ← DATA''' - testingData;
12         for each Learner do
13           classifier ← Learner(trainingData);
14           evaluate eight classification performance values of
             classifier on testingData;
15           end for
16         end for
17       end for
18       for each Learner do
19         Measure ← evaluate average performance of each
           processMetric;
20       end for
21     end for
22   end for
23 end for
24 End
```

As well as comparing the class correlation values and classification performance values, we conduct Wilcoxon matched-pair signed-rank test [50] with 95% confidence interval, a nonparametric test method for two or more related samples, to test whether the difference of class correlation values among five process metrics is significant and whether the classification performance difference of five process metrics is significant. This statistical method has been widely used in SDP [51], [52]. The original assumption is that there is no significant difference among five process metrics when the confidence interval is 95%. If the *P* value is below 0.05, the original hypothesis is rejected, that is, there is significant difference among five process metrics.

Even though the significance test results show that it has reached the significant level, if the effect size is too small, it also lacks practical value. So, in order to further illustrate the degree of different among five process metrics in terms of class correlation and classification performance, we also apply Cohen's d value [53] to calculate the difference between NDC and other process metrics. Cohen's d is the effect size, which is not affected by the number of samples. It is defined as

$$Cohen's\ d = \frac{\mu_1 - \mu_2}{\sqrt{(\sigma_1^2 + \sigma_2^2)/2}} \tag{14}$$

Where μ_1 and μ_2 represent the average value of each sample, and σ_1 and σ_2 represent the standard deviation. The effect size of Cohen's d can be divided into four levels: $Cohen's\ d < 0.2$ (Negligible, N), $0.2 \le Cohen's\ d < 0.5$ (Small, S), $0.5 \le Cohen's\ d < 0.8$ (Medium, M) and $Cohen's\ d \ge 0.8$ (Large, L).

IV. EXPERIMENTAL RESULTS AND ANALYSIS

In this section, we not only show the correlation between each process metric and the change of defect state, and the classification performance of each process metric in figures, but also perform statistical analysis on the class correlation values of each process metrics and the classification performance of each process metric respectively, to verify whether the experimental results are of practical value.

A. CORRELATION BETWEEN EACH PROCESS METRIC AND CLASSES

In this experiment, Pearson Correlation Coefficient, Chi-Square, ReliefF, Information Gain, Gain Ratio, and Symmetric Uncertainty are used to measure the correlation between each process metric and the change of defect state. Figure 2 shows the class correlation of five process metrics obtained by six class correlation measurement methods on seven open source software projects with 18 versions and five commercial projects with 19 versions respectively. The abscissa represents the projects in Table I, and the ordinate represents the class correlation values.

According to Figure 2, we can observe that NDC process metric can obtain the highest class correlation values among five process metrics in all class correlation measurement methods and almost all projects, followed by NR, and ANML has higher class correlation values compared with NML, whereas DCM obtains the lowest class correlation among five process metrics.

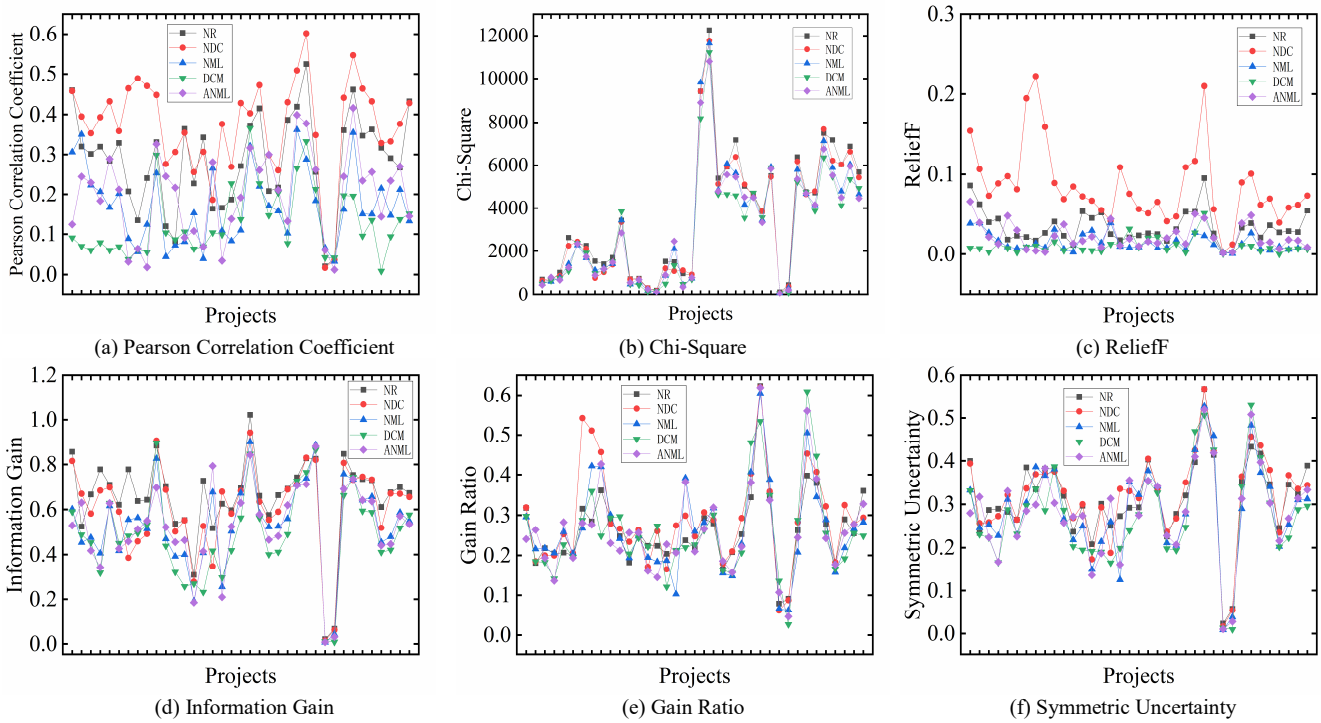
To further explore whether there is significant difference among NDC and other process metrics on the class correlation, the Wilcoxon matched-pair signed-rank test with 95% confidence interval is applied. If the *P* value is below 0.05, the original hypothesis is rejected, that is, NDC is significantly better than other process metrics. Table III shows the significance test results. In Table III, the bold *P* value is below 0.05, and "△" indicates that NDC is not superior to other process metric.

From Table III, we can observe that there is a significant difference among NDC and other process metrics for almost

all class correlation measurement methods.

In addition, to give a clearer comparison among NDC and other process metrics on the class correlation, we compare the effect size over all 37 datasets between NDC and other process metrics according to Cohen's d, and the Cohen's d result is shown in Table IV.

From Table IV, we can observe that compared with other process metrics, in almost all of the class correlation measurement methods, the class correlation of NDC can produce a certain effect size, even a large effect size.



(a) Pearson Correlation Coefficient (b) Chi-Square (c) ReliefF

(d) Information Gain (e) Gain Ratio (f) Symmetric Uncertainty

FIGURE 2. Class correlation values of five process metrics on six measurement methods

TABLE III. The significance test results of NDC and other process metrics on class correlation

Methods	NDC			
	NR	NML	DCM	ANML
Pearson Correlation Coefficient	**9.08E-7**	**2.02E-7**	**1.46E-7**	**3.55E-7**
Chi-Square	**△0.001**	**0.003**	**1.39E-4**	**1.09E-4**
ReliefF	**1.24E-7**	**1.24E-7**	**1.14E-7**	**1.34E-7**
Information Gain	**△8.20E-5**	**2.32E-4**	**1.00E-5**	**5.30E-5**
Gain Ratio	**0.002**	**0.022**	**0.006**	**0.031**
Symmetric Uncertainty	0.197	**2.53E-4**	**8.00E-6**	**3.29E-4**

TABLE IV. The Cohen's d results of NDC and other process metrics on class correlation

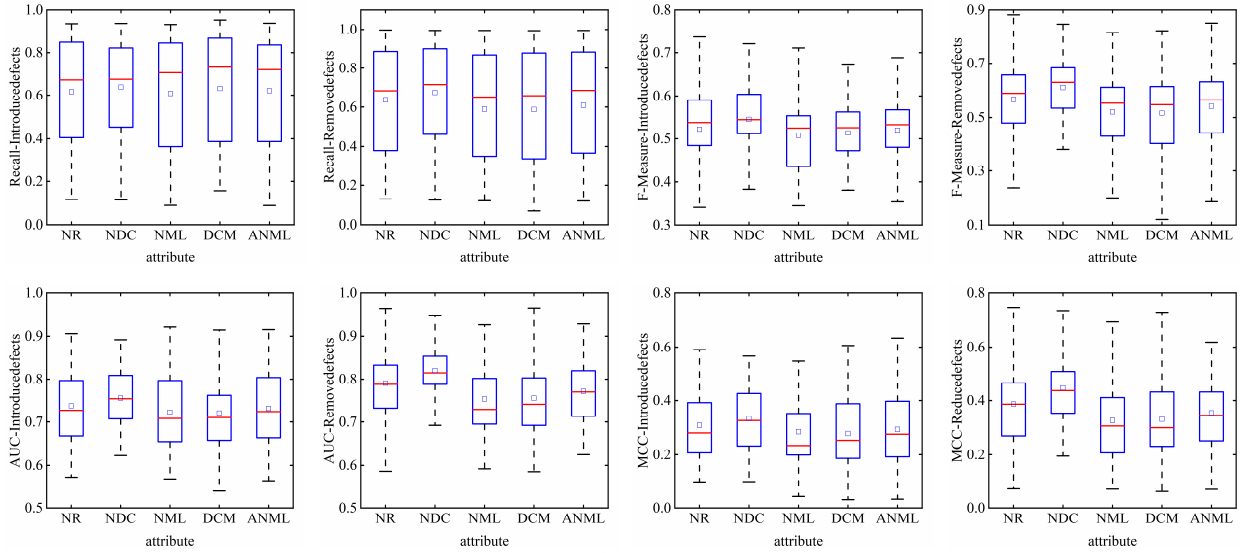
Methods	NDC			
	NR	NML	DCM	ANML
Pearson Correlation Coefficient	0.720 (M)	1.849 (L)	2.290 (L)	1.503 (L)
Chi-Square	-0.054 (NN)	0.073 (N)	0.168 (N)	0.119 (N)
ReliefF	1.380 (L)	1.995 (L)	2.117 (L)	1.741 (L)
Information Gain	-0.231 (NS)	0.375 (S)	0.580 (M)	0.422 (S)
Gain Ratio	0.256 (S)	0.218 (S)	0.271 (S)	0.230 (S)
Symmetric Uncertainty	0.044 (N)	0.286 (S)	0.424 (S)	0.323 (S)

In short, we conclude that for these six class correlation measurement methods, the correlation between NDC and the change of defect state is significantly higher than that between other process metrics and the change of defect state, and has a certain amount of effect NDC, and NR is the second, whereas DCM is the last. In addition, ANML is superior to NML.

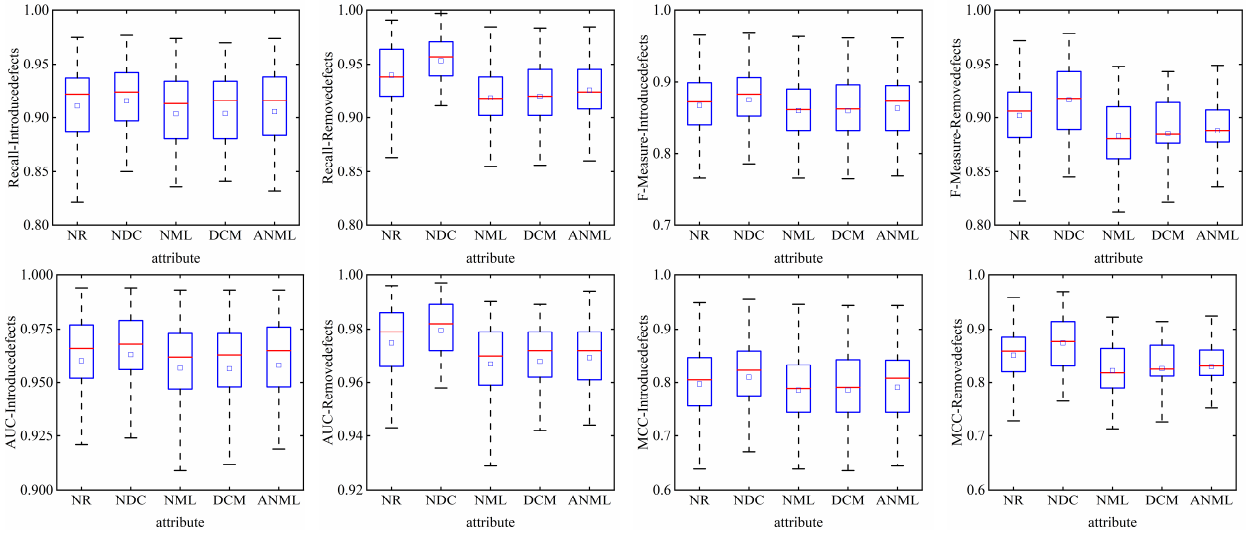
B. CLASSIFICAITON PERFORMANCE OF PROCESS METRICS

This experiment uses NB, KNN, LR, MLP, and SVM as classification algorithms, and all code metrics and each process metric as input variable, to build classification models, and then compares their performance, to study on which process metrics have better classification performance for the change of defect state. Boxplots on five classification algorithms are drawn to clearly and intuitively compare the performance of all process metrics on all projects respectively, as shown in Fig. 3 (a), (b), (c), (d), and (e). The abscissa represents five process metrics, and the ordinate represents the classification performance values. The red line represents the median of all performance measure values of each metric, and the blue square represents the average value of all performance measure values of each metric.

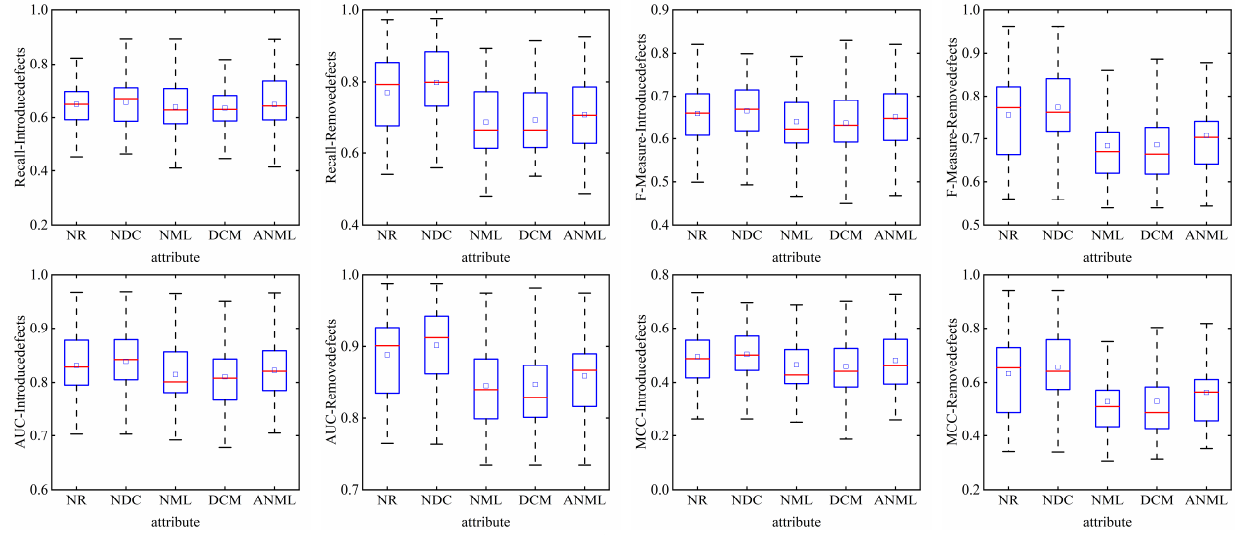
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2994528, IEEE Access



(a) NB

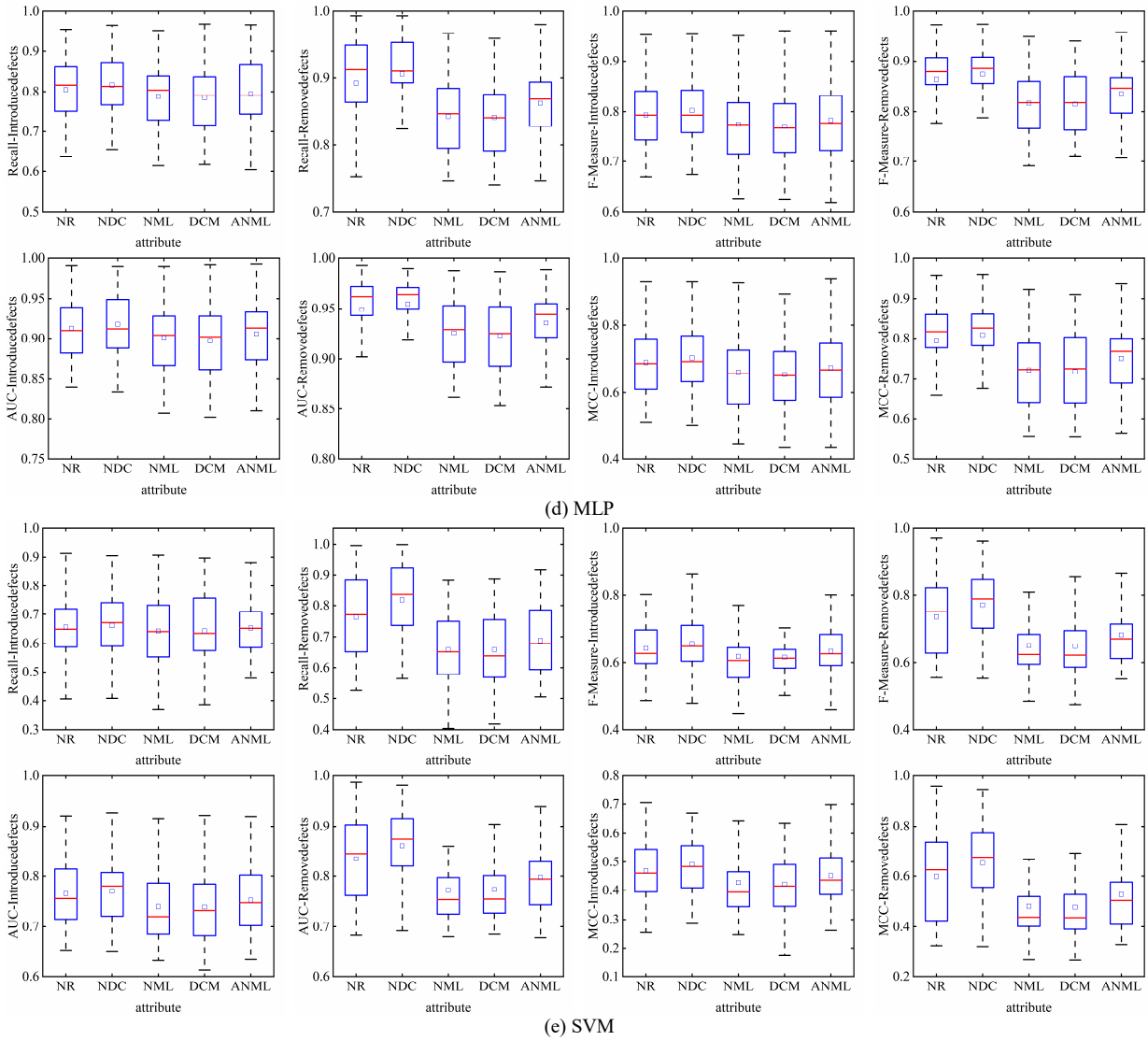


(b) KNN



(c) LR

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2994528, IEEE Access



(d) MLP

(e) SVM

FIGURE 3. Boxplots of eight evaluation measures of five process metrics obtained by five classification algorithms across 37 datasets with SMOTE

From Figure 3, we can find the following conclusions.

i) In terms of Recall, the performance values of five process metrics are similar, but according to the average values, NDC is the best.

ii) In terms of F-Measure, the classification performance of NDC is the best, followed by NR, and ANML ranks third.

iii) In terms of AUC, the performance values of five process metrics are all superior to the performance of random classification, which is 0.5. Further to say, the median and average AUC values of all process metrics are all above 0.7, and some even hit 0.9, which indicates that classification models can be accepted. Similarly, the classification performance of NDC is the best, followed by NR and ANML.

iv) In terms of MCC, the performance of NDC is the best, followed by NR and ANML. The values of process metrics are all above 0, indicating that the predicted classes are positively correlated with the actual classes. Further to say, the average performance values of process metrics are all above 0.2, which indicates that their classification performance values are good.

v) The classification performance of all process metrics for elimination of defects is better than that for introduction of defects, especially of NDC and NR.

vi) The advantage of NDC in elimination of defects is more obvious than that in introduction of defects.

So, we can conclude that the classification performance of all process metrics for elimination of defects is better than that for introduction of defects. And among five process metrics, NDC can obtain the best classification performance for the change of defect state in all evaluation measures, followed by NR, and ANML ranks third, which is better than NML. In addition, the advantage of NDC in elimination of defects is more obvious than that in introduction of defects.

To determine the statistical significance between the classification performance of NDC and that of other process metrics, the Wilcoxon matched-pair signed-rank test with 95% confidence interval is applied. Table V shows the significance test results of eight evaluation measures between NDC and others on five classification algorithms.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2994528, IEEE Access

TABLE V. The significance test results of the classification performance between NDC and others on NB, KNN, LR, MLP, and SVM with SMOTE

Classifier	introduction of defects	NDC				elimination of defects	NDC			
		NR	NML	DCM	ANML		NR	NML	DCM	ANML
NB	Recall	0.392	0.070	0.964	0.287	Recall	**3.70E-5**	**1.48E-4**	**0.001**	**0.004**
	F-Measure	**0.027**	**4.90E-4**	**0.001**	**0.006**	F-Measure	**1.70E-5**	**1.00E-6**	**2.00E-6**	**2.84E-4**
	AUC	**0.002**	**4.60E-5**	**6.00E-5**	**0.028**	AUC	**1.70E-5**	**6.40E-7**	**1.68E-7**	**3.20E-5**
	MCC	**0.010**	**1.48E-4**	**2.10E-5**	**0.002**	MCC	**3.20E-5**	**5.85E-7**	**3.41E-7**	**9.20E-5**
KNN	Recall	**0.005**	**2.90E-5**	**1.18E-4**	**5.00E-5**	Recall	**1.30E-5**	**2.16E-7**	**1.67E-7**	**1.28E-7**
	F-Measure	**8.00E-5**	**7.71E-7**	**5.00E-6**	**3.00E-6**	F-Measure	**3.00E-6**	**1.99E-7**	**2.45E-7**	**1.67E-7**
	AUC	**0.001**	**3.38E-7**	**3.00E-6**	**2.00E-6**	AUC	**2.20E-5**	**1.89E-7**	**1.81E-7**	**1.65E-7**
	MCC	**7.00E-5**	**8.66E-7**	**6.00E-6**	**3.00E-6**	MCC	**2.00E-6**	**1.46E-7**	**2.35E-7**	**1.24E-7**
LR	Recall	0.330	0.078	0.053	0.441	Recall	**0.009**	**2.57E-7**	**1.86E-7**	**2.00E-6**
	F-Measure	0.460	**1.96E-4**	**4.10E-5**	0.070	F-Measure	**0.047**	**2.19E-7**	**1.58E-7**	**4.00E-6**
	AUC	0.084	**1.40E-5**	**2.00E-6**	**0.005**	AUC	**0.011**	**2.28E-7**	**1.58E-7**	**4.00E-6**
	MCC	0.465	**3.00E-5**	**1.10E-5**	**0.038**	MCC	**0.048**	**2.02E-7**	**1.46E-7**	**8.00E-6**
MLP	Recall	**0.007**	**9.05E-7**	**7.16E-7**	**9.40E-5**	Recall	**0.018**	**3.02E-7**	**1.14E-7**	**1.64E-7**
	F-Measure	**0.011**	**5.03E-7**	**4.14E-7**	**3.50E-5**	F-Measure	**0.024**	**1.45E-7**	**1.23E-7**	**1.67E-7**
	AUC	**0.046**	**1.00E-6**	**4.13E-7**	**3.90E-5**	AUC	**0.019**	**2.18E-7**	**1.14E-7**	**3.42E-7**
	MCC	**0.012**	**4.66E-7**	**4.31E-7**	**3.10E-5**	MCC	**0.026**	**1.46E-7**	**1.24E-7**	**1.83E-7**
SVM	Recall	△0.874	0.278	0.281	0.519	Recall	**1.90E-4**	**3.84E-7**	**2.26E-7**	**2.00E-6**
	F-Measure	**0.048**	**3.10E-5**	**2.50E-5**	**0.004**	F-Measure	**0.001**	**1.86E-7**	**1.46E-7**	**4.86E-7**
	AUC	0.140	**4.30E-5**	**6.20E-5**	**0.014**	AUC	**0.004**	**3.15E-7**	**2.58E-7**	**1.00E-6**
	MCC	**0.036**	**2.00E-6**	**3.00E-6**	**0.003**	MCC	**0.001**	**1.87E-7**	**1.46E-7**	**4.50E-7**

From Table V, we can observe that there is a significant difference between the classification performance of NDC and that of other process metrics in most evaluation measures and all five classification algorithms, especially for elimination of defects.

In addition, to give a clearer comparison among NDC and other process metrics on the classification performance, we compare the effect size over all 37 datasets between NDC and other process metrics according to Cohen's d, and the Cohen's d result is shown in Table VI.

TABLE VI. The Cohen's d results of the classification performance between NDC and others on NB, KNN, LR, MLP, and SVM with SMOTE

Classifier	introduction of defects	NDC				elimination of defects	NDC			
		NR	NML	DCM	ANML		NR	NML	DCM	ANML
NB	Recall	0.090 (N)	0.121 (N)	0.026 (N)	0.070 (N)	Recall	0.145 (N)	0.319 (S)	0.318 (S)	0.249 (S)
	F-Measure	0.206 (S)	0.327 (S)	0.298 (S)	0.230 (S)	F-Measure	0.281 (S)	0.597 (M)	0.592 (M)	0.457 (S)
	AUC	0.240 (S)	0.418 (S)	0.448 (S)	0.307 (S)	AUC	0.352 (S)	0.796 (M)	0.768 (M)	0.573 (M)
	MCC	0.189 (N)	0.360 (S)	0.415 (S)	0.292 (S)	MCC	0.344 (S)	0.743 (M)	0.686 (M)	0.566 (M)
KNN	Recall	0.117 (N)	0.288 (S)	0.287 (S)	0.240 (S)	Recall	0.466 (S)	1.214 (L)	1.124 (L)	0.966 (L)
	F-Measure	0.161 (N)	0.305 (S)	0.307 (S)	0.240 (S)	F-Measure	0.419 (S)	0.994 (L)	0.951 (L)	0.829 (L)
	AUC	0.131 (N)	0.282 (S)	0.294 (S)	0.223 (S)	AUC	0.350 (S)	0.904 (L)	0.839 (L)	0.731 (M)
	MCC	0.161 (N)	0.304 (S)	0.306 (S)	0.238 (S)	MCC	0.421 (S)	0.992 (L)	0.949 (L)	0.828 (L)
LR	Recall	0.073 (N)	0.163 (N)	0.202 (S)	0.075 (N)	Recall	0.255 (S)	1.042 (L)	1.013 (L)	0.883 (L)
	F-Measure	0.077 (N)	0.285 (S)	0.321 (S)	0.158 (N)	F-Measure	0.198 (N)	0.997 (L)	0.986 (L)	0.771 (M)
	AUC	0.136 (N)	0.410 (S)	0.470 (S)	0.285 (S)	AUC	0.216 (S)	0.928 (L)	0.909 (L)	0.718 (M)
	MCC	0.081 (N)	0.319 (S)	0.359 (S)	0.184 (N)	MCC	0.162 (N)	0.899 (L)	0.898 (L)	0.673 (M)
MLP	Recall	0.146 (N)	0.349 (S)	0.386 (S)	0.270 (S)	Recall	0.196 (N)	0.961 (L)	1.037 (L)	0.687 (M)
	F-Measure	0.137 (N)	0.401 (S)	0.450 (S)	0.280 (S)	F-Measure	0.156 (N)	0.881 (L)	0.928 (L)	0.628 (M)
	AUC	0.130 (N)	0.410 (S)	0.498 (S)	0.299 (S)	AUC	0.169 (N)	0.857 (L)	0.920 (L)	0.604 (M)
	MCC	0.137 (N)	0.406 (S)	0.459 (S)	0.281 (S)	MCC	0.151 (N)	0.877 (L)	0.927 (L)	0.627 (M)
SVM	Recall	0.049 (N)	0.152 (N)	0.148 (N)	0.072 (N)	Recall	0.439 (S)	1.323 (L)	1.323 (L)	1.094 (L)
	F-Measure	0.164 (N)	0.442 (S)	0.477 (S)	0.269 (S)	F-Measure	0.323 (S)	1.201 (L)	1.240 (L)	0.887 (L)
	AUC	0.077 (N)	0.461 (S)	0.467 (S)	0.269 (S)	AUC	0.332 (S)	1.272 (L)	1.266 (L)	0.886 (L)
	MCC	0.193 (N)	0.530 (M)	0.584 (M)	0.336 (S)	MCC	0.322 (S)	1.158 (L)	1.198 (L)	0.839 (L)

From Table VI, we can observe that compared with other process metrics except NR, in almost all of the classification algorithms and evaluation measures, NDC can achieve a smaller or even larger effect on the classification performance. At the same time, compared with the introduction of defects, NDC has more effect on elimination of defects.

In conclusion, the classification performance of NDC is significantly better than other process metric in most evaluation measures, and can obtain a certain amount of effect. And NDC is more important for the elimination of defects than for the introduction of defects. Therefore, we suggest that when the number of defects is large, we should reduce the number of developers, whereas when the number of defects is small, the number of developers can be increased to improve the development efficiency. Moreover, the classification performance of ANML is better than that of NML. Therefore, we suggest that when predicting software defect, we should extract NDC and the code relative change metrics, not only NML.

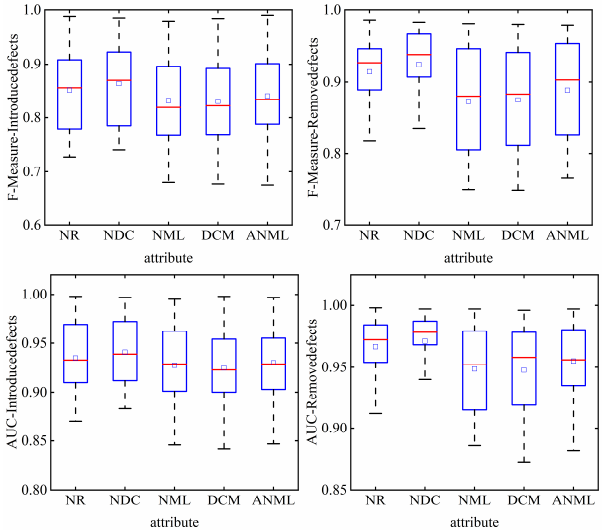
C. DISCUSSION ABOUT DIFFERENT CLASS IMBALANCE PROCESSING

In this study, the experimental datasets suffer from varying degrees of class imbalance problem, so we use SMOTE to handle the class imbalance problem of the experimental datasets before conducting the experiments. There are plenty of class imbalance handling methods, such as under-sampling and over-sampling. To avoid the influence of different class imbalance handling methods on the classification performance, we also compare the classification performance for the change of defects state

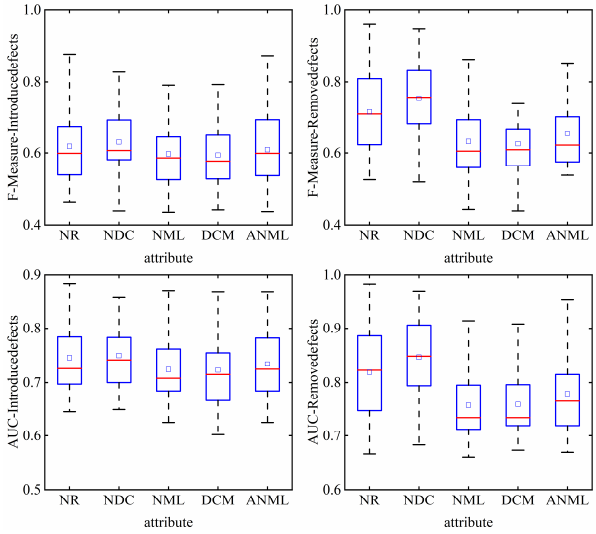
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2994528, IEEE Access

among five process metrics on under-sampling and over-sampling respectively. Figures 4 and 5 show the boxplots of F-Measure and AUC values of five process metrics obtained by MLP and SVM across 37 versions of 12 projects with over-sampling and under-sampling respectively.

From Figures 4 and 5, we can observe that the median and average of AUC values are all above 0.7, so we can accept these classification models. In addition, the classification performance of all process metrics for elimination of defects is better than that for introduction of defects. Moreover, NDC can obtain the best classification performance for the change of defect state among five process metrics, followed by NR, and ANML ranks third, which are better than NML. The advantage of NDC in elimination of defects is more obvious than that in introduction of defects. We also conduct experiments on other evaluation measures and other classifiers, and the same results have been obtained, which are not fully listed for space reasons. The conclusion is consistent with the datasets processed with SMOTE.

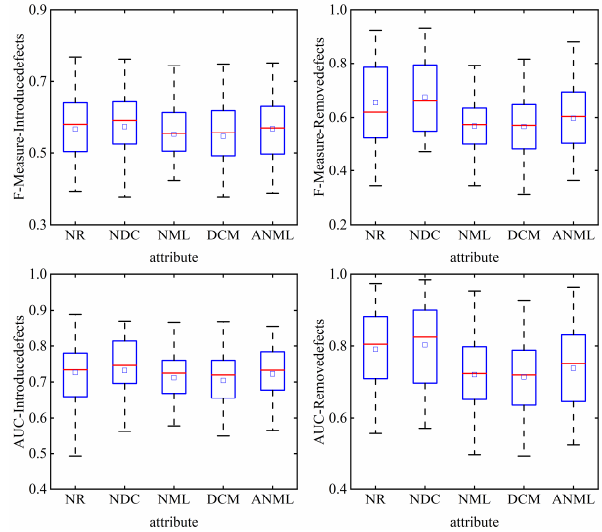


(a) MLP

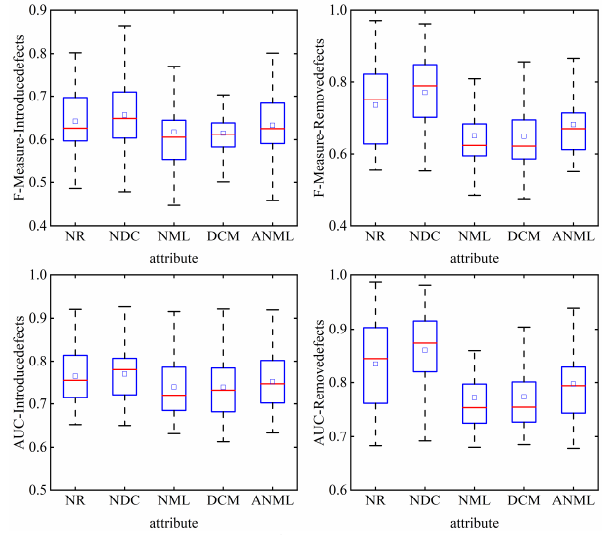


(b) SVM

FIGURE 4. Boxplots of F-Measure and AUC values of five process metrics obtained by MLP and SVM across 37 datasets with over-sampling



(a) MLP



(b) SVM

FIGURE 5. Boxplots of F-Measure and AUC values of five process metrics obtained by MLP and SVM across 37 datasets with under-sampling

We conduct the Wilcoxon matched-pair signed-rank test at a confidence level of 95%. Tables VII and VIII show the significance test results in terms of over-sampling and under-sampling respectively. We also compare the effect size over all 37 datasets with over-sampling and under-sampling between NDC and other process metrics according to Cohen's d, and the Cohen's d results of over-sampling and under-sampling are shown in Tables IX and X respectively.

From Tables VII to X, we can observe that NDC generally achieves the best classification results. And there is a significant difference between the classification performance of NDC and that of other process metrics in most evaluation measures with over-sampling and under-sampling, and NDC can obtain a smaller or even larger effect on the classification performance. We also conduct the Wilcoxon matched-pair signed-rank test and Cohen's d on other evaluation measures and other classifiers, and the same results have been obtained, which are not fully listed for space reasons. So, we can

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2994528, IEEE Access

conclude that the experiment results with other class imbalance handling methods is consistent with SMOTE, and class imbalance handling method has no effect on the experimental results.

TABLE VII. The significance test results of F-Measure and AUC between NDC and others on MLP and SVM over 37 datasets with over-sampling

Classifier	introduction of defects	NDC				elimination of defects	NDC			
		NR	NML	DCM	ANML		NR	NML	DCM	ANML
MLP	F-Measure	**0.001**	**7.18E-7**	**7.77E-7**	**3.00E-6**	F-Measure	**0.002**	**3.68E-7**	**3.27E-7**	**8.02E-7**
	AUC	**0.010**	**2.30E-5**	**5.00E-6**	**6.50E-5**	AUC	**0.001**	**4.00E-6**	**8.70E-7**	**6.00E-6**
SVM	F-Measure	0.103	**0.001**	**3.02E-4**	**0.015**	F-Measure	**0.001**	**1.72E-7**	**1.24E-7**	**1.72E-7**
	AUC	0.122	**1.15E-4**	**2.77E-4**	**0.003**	AUC	**3.73E-4**	**1.58E-7**	**1.71E-7**	**3.02E-7**

TABLE VIII. The significance test results of F-Measure and AUC between NDC and others on MLP and SVM over 37 datasets with under-sampling

Classifier	introduction of defects	NDC				elimination of defects	NDC			
		NR	NML	DCM	ANML		NR	NML	DCM	ANML
MLP	F-Measure	0.769	**0.016**	**0.002**	0.365	F-Measure	**0.021**	**1.72E-7**	**1.65E-7**	**1.00E-6**
	AUC	0.850	**0.020**	**0.001**	0.169	AUC	0.087	**3.55E-7**	**3.02E-7**	**3.57E-7**
SVM	F-Measure	**0.048**	**3.10E-5**	**2.50E-5**	**0.004**	F-Measure	**0.001**	**1.86E-7**	**1.46E-7**	**4.86E-7**
	AUC	0.140	**4.30E-5**	**6.20E-5**	**0.014**	AUC	**0.004**	**3.15E-7**	**2.58E-7**	**1.00E-6**

TABLE IX. The Cohen's d results of F-Measure and AUC between NDC and others on MLP and SVM over 37 datasets with over-sampling

Classifier	introduction of defects	NDC				elimination of defects	NDC			
		NR	NML	DCM	ANML		NR	NML	DCM	ANML
MLP	F-Measure	0.168 (N)	0.413 (S)	0.423 (S)	0.301 (S)	F-Measure	0.188 (N)	0.788 (M)	0.757 (M)	0.595 (M)
	AUC	0.171 (N)	0.349 (S)	0.414 (S)	0.288 (S)	AUC	0.182 (N)	0.747 (M)	0.746 (M)	0.584 (M)
SVM	F-Measure	0.119 (N)	0.337 (S)	0.366 (S)	0.217 (S)	F-Measure	0.347 (S)	1.213 (L)	1.315 (L)	0.992 (L)
	AUC	0.058 (N)	0.368 (S)	0.379 (S)	0.220 (S)	AUC	0.365 (S)	1.288 (L)	1.308 (L)	0.955 (L)

TABLE X. The Cohen's d results of F-Measure and AUC between NDC and others on MLP and SVM over 37 datasets with under-sampling

Classifier	introduction of defects	NDC				elimination of defects	NDC			
		NR	NML	DCM	ANML		NR	NML	DCM	ANML
MLP	F-Measure	0.060 (N)	0.204 (S)	0.247 (S)	0.050 (N)	F-Measure	0.153 (N)	0.856 (L)	0.882 (L)	0.606 (M)
	AUC	0.057 (N)	0.196 (N)	0.262 (S)	0.093 (N)	AUC	0.105 (N)	0.749 (M)	0.811 (L)	0.585 (M)
SVM	F-Measure	0.164 (N)	0.442 (S)	0.477 (S)	0.269 (S)	F-Measure	0.323 (S)	1.201 (L)	1.240 (L)	0.887 (L)
	AUC	0.077 (N)	0.461 (S)	0.467 (S)	0.269 (S)	AUC	0.332 (S)	1.272 (L)	1.266 (L)	0.886 (L)

V. THREATS TO VALIDITY

In this section, we describe the threats to validity of our study in construct validity, internal validity, external validity, and conclusion validity.

A. THREATS TO CONSTRUCT VALIDITY

In this paper, we use Recall, F-Measure, AUC, and MCC to present the classification performance. These results can be further refined by using other performance evaluation measures.

Also, we use the jar package provided by Weka tool to implement our study. Weka is a tool used for machine learning and data mining. We believe that Weka is reliable.

B. THREATS TO INTERNAL VALIDITY

In this paper, most popular and widely used algorithms were considered. We select six class correlation measurement methods and five classical classification algorithms to conduct the experiments. In fact, there are other class correlation measurement methods and classification algorithms. Comparison can be done among a greater number of algorithms. We will include more techniques in the comparative analysis to provide new results in the future.

Also, we focus on five classical, widely used, and defects-related process metrics. There are other process metrics, such as the age of class files and experience of developers. Comparison can be done among other process metrics to get more useful conclusions.

C. THREATS TO EXTERNAL VALIDITY

The seven open source projects and five commercial projects used in our study are all Java projects, and are different in application fields, scales, and the proportion of introduction of defects and elimination of defects. Consequently, our empirical study is universal, and the conclusions are general. These results can be further refined by using a greater number of datasets. Increased number of datasets will strengthen the experimental results.

D. THREATS TO CONCLUSION VALIDITY

In the second experiment, in order to eliminate the effect of randomly dividing the instances, we performed 10 times 10-fold cross validation. In addition, Wilcoxon matched-pair signed-rank test and Cohen's d are used to test whether the experimental result among five process metrics is statistically significant and calculate the effect size.

VI. CONCLUSION

Software defect prediction is the application of machine learning in software engineering. In this paper, we focus on the change of defect state of software modules, including introduction of defects and elimination of defects. We investigate which process metrics are significantly important to change of defects in evolving projects by conducting an empirical study on 18 release versions of seven open source

projects and 19 release versions of five commercial projects. In detail, we compare the class correlation values among five process metrics by using six class correlation measurement methods, and the classification performance values among five process metrics in terms of four evaluation measures by using five classification algorithms. We also perform statistical analysis of Wilcoxon matched-pair signed-rank test and Cohen's d to verify whether the experimental results are statistically significant and calculate the effect size. The experimental results indicate that among these five process metrics, Number of Distinct Committers (NDC) plays a significantly important role in the change of defect state, especially for elimination of defects, and Number of Revisions (NR) is the second, whereas Degree of Code Modification (DCM) is the last. In addition, Average Number of Modified Lines (ANML) is superior to Number of Modified Lines (NML). Based on the experimental results, some suggestions for software development and software defect prediction are also discussed. We suggest that when the number of defects is large, software development managers should reduce the number of developers, whereas when the number of defects is small, the number of developers can be increased to improve the development efficiency. Moreover, ANML is more related to the introduction of defects and the elimination of defects than NML, and the classification performance of ANML is better than that of NML. Therefore, we suggest that when predicting software defect, we should extract the code relative change metrics as well as NDC, not only NML.

In the future, comparison can be done among other process metrics by using more datasets, class correlation measurement methods, classification algorithms, and performance evaluation measures to get more useful conclusions of software development and software testing.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers and editors for suggesting improvements and for their very helpful comments. Special thanks to all the individuals who participated and contributed to improve the quality and readability of this paper.

REFERENCES

[1] N. E. Fenton and N. Ohlsson, "Quantitative analysis of faults and failures in a complex software system," *IEEE Trans. Software Eng.*, vol. 26, no. 8, pp. 797-814, Aug. 2002.

[2] J. C. Knight, "Safety critical systems: Challenges and directions," in *Proc. 24th Int. Conf. Softw. Eng.*, Orlando, FL, USA, 2002, pp. 547-550.

[3] Z. Li, X. Jing, and X. Zhu, "Progress on approaches to software defect prediction," *IET Softw.*, vol. 12, no. 3, pp. 161-175, Jun. 2018.

[4] S. Lessmann, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *IEEE Trans. Software Eng.*, vol. 34, no. 4, pp. 485-496, Jul. 2008.

[5] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, "Predicting the location and number of faults in large software systems," *IEEE Trans. Software Eng.*, vol. 31, no. 4, pp. 340-355, Apr. 2005.

[6] T. J. McCabe, "A complexity measure," *IEEE Trans. Software Eng.*, vol. SE-2, no. 4, pp. 308-320, 1976.

[7] M. H. Halstead, "Elements of software science," Elements of software science, New York, USA: Elsevier, 1978.

[8] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Trans. Software Eng.*, vol. 20, no. 6, pp. 476-493, Jun. 1994.

[9] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for Eclipse," in *Proc. 3rd Int. Workshop Predictor Models Softw. Eng.*, Minneapolis, MN, USA, 2007.

[10] H. M. Olague, L. H. Etzkorn, S. Gholston, and S. Quattlebaum, "Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes," *IEEE Trans. Software Eng.*, vol. 33, no. 6, pp. 402-419, Jun. 2007.

[11] M. Shepperd and D. C. Ince, "A critique of three metrics," *J. Syst. Softw.* (*JSS*), vol. 26, no. 3, pp. 197-210, Sept. 1994.

[12] M. Jureczko and L. Madeyski, "Software product metrics used to build defect prediction models," Rev. 2, 2014. [Online]. Available: http://madeyski.e-informatyka.pl/download/JureczkoMadeyskiSoftwareProductMetrics.pdf.

[13] D. Radjenović, M. Heričko, R. Torkar, and A. Živkovič, "Software fault prediction metrics: A systematic literature review," *Inf. Softw. Technol.*, vol. 55, no.8, pp. 1397-1418, Aug. 2013.

[14] F. Rahman and P. Devanbu, "How, and why, process metrics are better," in *Proc. 35th Int. Conf. Softw. Eng.* (*ICSE*), San Francisco, CA, USA, 2013, pp. 432-441.

[15] A. Schröter, T. Zimmermann, R. Premraj, and A. Zeller, "If your bug database could talk," in *Proc. 5th Int. Symp. Empirical Softw. Eng.*, Rio de Janeiro, Brazil, vol. 2, 2006, pp. 18-20.

[16] Y. Wu, Y. Yang, Y. Zhao, H. Lu, Y. Zhou, and B. Xu, "The influence of developer quality on software fault-proneness prediction," in *Proc. 8th Int. Conf. Softw. Security Rel.*, San Francisco, CA, USA, 2014, pp. 11-19.

[17] N. Nagappan, B. Murphy, and V. Basili, "The influence of organizational structure on software quality: An empirical case study," in *Proc. 30th Int. Conf. Softw. Eng.*, Leipzig, Germany, 2008, pp. 521-530.

[18] A. Mockus, "Organizational volatility and its effects on software defects," in *Proc. ACM SIGSOFT 18th Int. Symp. Found. Softw. Eng.*, Santa Fe, New Mexico, USA, 2010, pp. 117-126.

[19] L. Madeyski and M. Jureczko, "Which process metrics can significantly improve defect prediction models? An empirical study," *Softw. Qual. J.*, vol. 23, no. 3, pp. 393-422, Sept. 2015.

[20] R. Moser, W. Pedrycz, and G. Succi, "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," in *Proc. 30th Int. Conf. Softw. Eng.*, Leipzig, Germany, 2008, pp. 181-190.

[21] A. E. Hassan, "Predicting faults using the complexity of code changes," in *Proc. 31st Int. Conf. Softw. Eng.*, Vancouver, BC, Canada, 2009, pp. 78-88.

[22] T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy, "Predicting fault incidence using software change history," *IEEE Trans. Software Eng.*, vol. 26, no. 7, pp. 653-661, Jul. 2000.

[23] B. Stanić and W. Afzal, "Process metrics are not bad predictors of fault proneness," in *Proc. IEEE Int. Conf. Softw. Qual. Rel. Security Companion* (*QRS-C*), Prague, Czech Republic, 2017, pp. 493-499.

[24] E. J. Weyuker, T. J. Ostrand, and R. M. Bell, "Do too many cooks spoil the broth? Using the number of developers to enhance defect prediction models," *Empir. Softw. Eng.*, vol. 13, no. 5, pp. 539-559, Oct. 2008.

[25] T. Illes-Seifert and B. Paech, "Exploring the relationship of a file's history and its fault-proneness: An empirical study," in *Proc. TAIC-PART*, Windsor, UK, 2008, pp. 13-22.

[26] S. Matsumoto, Y. Kamei, A. Monden, K. Matsumoto, and M. Nakamura, "An analysis of developer metrics for fault prediction," in *Proc. 6th Int. Conf. Predictive Models Softw. Eng.*, Timişoara, Romania, 2010.

[27] N. Nagappan and T. Ball, "Use of relative code churn measures to predict system defect density," in *Proc. 27th Int. Conf. Softw. Eng.*, Saint Louis, MO, USA, 2005, pp. 284-292.

[28] N. Nagappan and T. Ball, "Using software dependencies and churn metrics to predict field failures: An empirical case study," in *Proc. 1st Int. Symp. Empir. Softw. Eng. Meas.* (*ESEM*), Madrid, Spain, 2007, pp. 364-373.

[29] Y. Shin, A. Meneely, L. Williams, and J. A. Osborne, "Evaluating

complexity, code churn, and developer activity metrics as indicators of software vulnerabilities," *IEEE Trans. Software Eng.*, vol. 37, no. 6, pp. 772-787, Nov. 2011.
[30] G. R. Choudhary, S. Kumar, K. Kumar, A. Mishra, and C. Catal, "Empirical analysis of change metrics for software fault prediction," *Comput. Electr. Eng.*, vol. 67, pp. 15-24, Apr. 2018.
[31] J. Liu, Y. Zhou, Y. Yang, H. Lu, and B. Xu, "Code churn: A neglected metric in effort-aware just-in-time defect prediction," in Proc. *2017 ACM/IEEE Int. Symp. Empir. Softw. Eng. Meas.* (*ESEM*), Toronto, ON, Canada, 2017, pp. 11-19.
[32] Q. Huang, X. Xia, and D. Lo, "Revisiting supervised and unsupervised models for effort-aware just-in-time defect prediction," *Empir. Softw. Eng.*, vol. 24, pp. 2823-2862, Oct. 2019.
[33] M. Miletić, M. Vukušić, G. Mauša, and T. G. Grbac, "Cross-release code churn impact on effort-aware software defect prediction," in *Proc. 41st Int. Conv. Inf. Commun. Technol., Electron. Microelectron.* (*MIPRO*), Opatija, Croatia, 2018, pp. 1460-1466.
[34] S. O. Kini and A. Tosun, "Periodic developer metrics in software defect prediction," in *Proc. 18th IEEE Int. Working Conf. Source Code Anal. Manip.* (*SCAM*), Madrid, Spain, 2018, pp. 72-81.
[35] J. C. Munson and S. G. Elbaum, "Code churn: A measure for estimating the impact of code change," in *Proc. Int. Conf. Softw. Maintenance*, Bethesda, MD, USA, 1998.
[36] G. H. John and P. Langley, "Estimating continuous distributions in Bayesian classifiers," in *Proc. 11th UAI*, Montréal, Qué, Canada, 1995, pp. 338-345.
[37] D. W. Aha, D. F. Kibler, and M. K. Albert, "Instance-based learning algorithms," *Mach. Learn.*, vol. 6, no. 1, pp. 37-66, Jan. 1991.
[38] S. L. Cessie and J. C. V. Houwelingen, "Ridge estimators in logistic regression," *J. R. Stat. Soc.*, vol. 41, no. 1, pp. 191-201, Jan. 1992.
[39] J. Platt, "Fast training of support vector machines using sequential minimal optimization," Advances in kernel methods: Support vector learning, New York, USA: ACM, 1999, pp.185-208.
[40] J. G. Attali and G. Pagès, "Approximations of functions by a multilayer perceptron: A new approach," *Neural Networks*, vol. 10, no. 6, pp. 1069-1081, Aug. 1997.
[41] X. Yang and W. Wen, "Ridge and Lasso Regression models for cross-version defect prediction," *IEEE Trans. Reliab.*, vol. 67, no. 3, pp. 885-896, Sept. 2018.
[42] S. Shukla, T. Radhakrishnan, K. Muthukumaran, and L. B. M. Neti, "Multi-objective cross-version defect prediction," *Soft Comput.*, vol. 22, pp. 1959-1980, Mar. 2018.
[43] S. D. Martino, F. Ferrucci, C. Gravino, and F. Sarro, "A genetic algorithm to configure support vector machines for predicting fault-prone components," in *Proc. 12th Int. Conf. Product-Focused Softw. Process Improvement*. Torre Canne, Italy, 2011, pp. 247-261.
[44] Y. Liu, Y. Li, J. Guo, Y. Zhou, B. Xu, "Connecting software metrics across versions to predict defects," in *Proc. 25th Int. Conf. Softw. Anal. Evol. Reengineering* (*SANER*), Campobasso, Italy, 2017, pp. 232-243.
[45] S. S. Rathore and S. Kumar, "An approach for the prediction of number of software faults based on the dynamic selection of learning techniques," *IEEE Trans. Reliab.*, vol. 68, no. 1, pp. 216-236, Mar. 2019.
[46] H. Lu, E. Kocaguneli, and B. Cukic, "Defect prediction between software versions with active learning and dimensionality reduction," in *Proc. 25th Int. Symp. Softw. Rel. Eng.*, Naples, Italy, 2014, pp. 312-322.
[47] Z. Xu, J. Liu, X. Luo, and T. Zhang, "Cross-version defect prediction via hybrid active learning with kernel principal component analysis," in *Proc. 25th Int. Conf. Softw. Anal. Evol. Reengineering* (*ISSTA*), Campobasso, Italy, 2017, pp. 209-220.
[48] I. H. Witten and E. Frank, "Data mining: Practical machine learning tools and techniques," 3rd ed. San Francisco, CA, USA: Morgan Kaufmann, 2011.
[49] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *J. Artif. Intell. Res.*, vol. 16, no. 1, pp. 321-357, Jun. 2002.
[50] G. Shieh, S. L. Jan, and R.H. Randles, "Power and sample size Determinations for the Wilcoxon signed-rank test," *J. Stat. Comput. Simul.*, vol.77, no.8, pp.717-724, Aug. 2007.
[51] Q. Yu, S. Jiang, and Y. Zhang, "The performance stability of defect prediction models with class imbalance: An empirical study," *IEICE Trans. Inf. Syst.*, vol. E100-D, no. 2, pp. 265-272, Feb. 2017.
[52] L. Gong, S. Jiang, L. Bo, L. Jiang, and J. Qian, "A novel class-imbalance learning approach for both within-project and cross-project defect prediction," *IEEE Trans. Reliab.*, vol. 69, no. 1, pp. 40-54, Mar. 2020.
[53] J. Cohen, "Statistical power analysis for the behavioral sciences," 2nd ed. Mahwah, NJ: Lawrence Erlbaum Associates, 1988.

LI JIANG is currently pursuing the M.S. degree with the School of Computer Science and Technology, China University of Mining and Technology. Her research interests include software analysis and testing, and machine learning.

SHUJUAN JIANG received the Ph.D. degree from Southeast University, in 2006. She was a Visiting Scholar with the Georgia Institute of Technology, from September 2008 to April 2009. She is currently a Professor and a Supervisor of the Ph.D. degree with the School of Computer Science and Technology, China University of Mining and Technology. Her research interests include compilation techniques and software engineering.

LINA GONG is currently pursuing the Ph.D. degree with the School of Computer Science and Technology, China University of Mining and Technology. Her research interests include software analysis and testing, and machine learning.

YUE DONG is currently pursuing the B.S. degree with SunYueqi Honors College, China University of Mining and Technology. His research interests include engineering of surveying and mapping.

QIAO YU received the Ph.D. degree from China University of Mining and Technology in 2017. She is a lecturer at School of Computer Science and Technology, Jiangsu Normal University. Her research interests include software analysis and testing, machine learning.