Technical Report
Number: NU-EECS-16-14

October, 2016

New findings on the use of static code attributes for defect prediction

Muhammed Maruf Öztürk, Ahmet Zengin

Abstract

Defect prediction includes tasks that are based on methods gener ated using software fault data sets and requires much effort to be completed. In defect prediction, although there are methods to conduct an analysis involving the classification of data sets and localisation of defects, those methods are not sufficient without eliminating repeated data points. The NASA Metrics Data Program (Nasa MDP) and Software Research Laboratory (SOFTLAP) data sets are frequently used in this field. Here, we present a novel method developed on the Nasa MDP and SOFTLAB data sets that detects repeated data points and analyses low level metrics. Also, a framework and an algorithm are presented for the proposed method. Statistical methods have been used for detecting repeated data points. This work sheds new lights on the extent to which repeated data adversely affects defect prediction performance, and stresses the importance of using low level metrics.

Keywords

Defect prediction · Software metrics · Software quality · Machine learning

Noname manuscript No.
(will be inserted by the editor)

New findings on the use of static code attributes for defect prediction

Muhammed Maruf Öztürk · Ahmet Zengin

Received: date / Accepted: date

Abstract Defect prediction includes tasks that are based on methods generated using software fault data sets and requires much effort to be completed. In defect prediction, although there are methods to conduct an analysis involving the classification of data sets and localisation of defects, those methods are not sufficient without eliminating repeated data points. The NASA Metrics Data Program (Nasa MDP) and Software Research Laboratory (SOFTLAP) data sets are frequently used in this field. Here, we present a novel method developed on the Nasa MDP and SOFTLAB data sets that detects repeated data points and analyses low level metrics. Also, a framework and an algorithm are presented for the proposed method. Statistical methods have been used for detecting repeated data points. This work sheds new lights on the extent to which repeated data adversely affects defect prediction performance, and stresses the importance of using low level metrics.

Keywords Defect prediction · Software metrics · Software quality · Machine learning

1 Introduction

In the software development process, 50 percent of the development time is spent on software testing. Defect prediction is generally used for reduction of the allocated time for testing and effort. Defect severity and defect count are basic terms for applying methods developed for defect prediction on source codes. If the budget required for testing processes is not sufficient, testers should focus on a specific part of the system. If the specific part tends to be more defect-prone than other parts of the software, it is an indicator of a

Department of Computer Engineering
Faculty of Computer and Information Science, Sakarya University
54187, Sakarya, Turkey Tel.: +90-2642956909
E-mail: muhammedozturk@sakarya.edu.tr · Science Institute of Sakarya University, Turkey

well-distributed budget. Therefore detecting defect-prone parts accurately is an important aspect that needs to be checked in detail and researched (Zimmermann *et al.*, 2009; Elish and Elish, 2008; Succi *et al.*, 2003). Most works in literature include remaining defects, relations of defects, and classifying defect-prone parts (Song *et al.*, 2011); however, in this work, we dealt the approach based on the classification of defects.

To classify software modules, some notions such as defect-prone and not defect-prone are commonly determined. Static code metrics are decisive factors while determining these properties. A reasonable data quality is required for successful defect prediction (Shepperd *et al.*, 2013; Liebchen and Shepperd, 2008). However, it is difficult to acquire fault data sets. Commercial software developers generally do not have any report of defect detection results. In addition, such results are generally not shared with public, for obvious reasons. It is acceptable to hide results if the defect severity is close to level of high; however, if these results are shared with researchers, there is a chance to prevent these defects.

Open source systems are frequently used by researchers to generate defect data sets (Rajesh Vasa and Jones, 2010; Keivanloo *et al.*, 2012; Linstead *et al.*, 2009). One of the issues is whether defect information is correct and consistent. In this respect, the main reasons for the encountered problems include inadequate documentation before preprocessing, showing little effort to improve skewed data while applying machine learning methods, and lack of reporting (Czibula *et al.*, 2014). As these processes require to human intervention, they are time consuming, and it is difficult to obtain defect data information from these data sets. Sometimes, this task might be impossible to accomplish precisely due to project sizes and flow speeds of defect information. After the verification of defect information, defect data are entered in to the version control comments. Thus, defect data become more reliable than originally.

The NASA PROMISEDATA and SOFTLAB data sets are commonly chosen for defect prediction. This is the reason we chose these data sets in this study (Menzies, 2015). In addition, our inspiring paper (Gray, 2013) clearly depicted that these data sets need further investigation to figure out the effects of repeated data points in classification. Although repositories include various data sets that are generated using a specific layout, they do not require a preprocessing prior to the its analysis. The use of detailed metrics alleviates the investigation of a software system. When researchers use NASA MDP data sets, it is assumed that these data sets have a certain quality level. However, new data acquired during the preprocessing stage can cause an undesirable result (Herzig *et al.*, 2013). At this point, the proposed framework and algorithm draw attention to data quality in classification problems and encourage researchers to verify data quality level.

The NASA and SOFTLAB data sets used in the experiment were obtained from software modules. Modules refer to software notions, such as class, function and procedure. The question that should be asked here is whether available metrics are sufficient for defect prediction. Though available metrics are sufficient for defect prediction, hiding source codes causes unverified codes that complicate understanding defect prediction results. To extract static code attributes, LOC (line of Code), Halstead, and McCabe are widely used standards (Halstead, 1977), (McCabe, 1976). Due to the similar metrics of SOFTLAB and NASA data sets, we used these data sets with combined. It is not a must to process data that are generated using same metric tables. However, it is easier to handle with the data that are generated using same metric tables. For instance the pc4 data set obtained from the NASA MDP repository includes the ar1 data set obtained from SOFTLAB, however the pc4 data set has additional metrics, such as Essential_Complexity and Percent_Comments. Thus, we extended the experimental data sets, and the data sets were suitable for the developed algorithm. The data sets are also suitable for binary classification. While classifying part of the data sets that are divided into training and testing, an effort was made to increase correct labelling of the data. However, a major shortcoming of labelling is disregarding defect severity.

Most works dealing with defect prediction take data quality issues into account to obtain reliable prediction results. In order to cope with such cases, it is desirable to detect noisy instances precisely (Verma and Gupta, 2012; Kim *et al.*, 2011; Hall *et al.*, 2012). Thus, defect data sets need to be eliminated regardless of their metric types are either static code metrics or process metrics (Bell *et al.*, 2013). Despite the fact that process metrics yielded promising results in recent years (He *et al.*, 2015; Madeyski and Jureczko, 2015), there is a strong need to develop defect prediction methods which use static code metrics (Madeyski and Jureczko, 2015). Defect prediction methods employing static code attributes have yet to be fully explored.

Our base research is Gray et al.'s work to compare with our study in terms of contributing defect prediction literature on the basis of data quality (Gray *et al.*, 2012; Gray, 2013). Their approach is to develop a novel cleansing method to reduce repeated data points in 15 NASA data sets (Menzies, 2015). These data sets retrieved from promise repository are publicly available. The method was employed successfully which means that repeated data can have a significant impact on the performance of classifiers including Bayes, naive Bayes, random forest, and j48. One of the challenges in detecting repeated data is that NASA data sets are not based on open-source. So it is difficult to validate the reality of repeated data points. Further, metrics describing each software module can be expressed with simple equations that are constituted with other metrics.

To overcome this problem, low level metrics have potentially favourable effects on detecting repeated data points in defect data sets that leads to achievement of remarkable classification performance. This motivates us to derive low level metrics and apply them to NASA data sets. We can list the reasons why this work was chosen for comparing as: focus on pre-processing instead of machine learning algorithms and using NASA MDP data sets. The effects of using low level metrics and repeated data point analysis are as yet unknown. This can be regarded as our motivation throughout the paper. Due to the similar metrics of SOFTLAB and NASA data sets, we used these data sets with combined. It is not a must to process data that are generated using same metric tables. However, it is easier to handle with the data that are generated using same metric tables.

The contributions of the paper can be summarised as follows: 1) a novel framework including a performance evaluation module, preprocessing module and metric extraction module, 2) determining whether repeated data points are general in all data sets, 3) observing the effect of repeated data points in classification, 4) evaluation of the effect of prediction accuracy using low level metrics.

The rest of the paper is organised as follows. Section 2 includes similar works and stresses the difference of our work. The framework and proposed preprocessing algorithm is explained in Section 3. Section 4 provides information regarding how the prediction performance is evaluated and which methods are generally used. In addition, experimental design and performance parameters of proposed methods are seen in Section 4. The comparison of our work and Gray's work is detailed in Section 5, and Section 6 includes conclusion and future works.

2 Background

2.1 Defect Prediction

The first step in predicting defects consists of determining which historical data to use and choosing the size for each data set. Note that unless the first step is completed, other processes should not be performed. The key factor is which data should be chosen for defect prediction. This depends on the works that will be compared with our work. However, initially a learning scheme must be generated. The structure of this scheme changes depending on the metric tables utilized. Further, the selection of public data is a fundamental step while using a learning scheme. Thus, the success of the learning method can be accurately evaluated.

2.2 Related Works

Various works related to defect prediction are available in the literature. One of them is Fenton et al.'s work (Fenton and Neil, 1999). In this work, defect prediction is handled with Bayes statistical methods. Twenty-two classification algorithms were applied with NASA MDP data sets, consequently linear models such as LogReg and LP produced similar results (Wang and Yao, 2013). In another work including class imbalance (Wang and Yao, 2013) five different class-imbalance learning methods were applied to ten data sets that AdaBoost.NC produced best results. However, using C4.5 demonstrates that further investigation is required. Another issue is the effect of developers on generation of defects. In Menzies and Koru's work published in 2011 (Menzies and Koru, 2013) it is illustrated that it is useful to know how many developers are on the revised files, it is not important to know which developer made these changes. The ship of the system access information of developers with defect severity was investigated in another work (Weyuker *et al.*, 2008), and this work also stressed that detecting high level defects using an automated test tool results in new directions to defect prediction works.

In Li et al.'s work (Li *et al.*, 2012) which used sample-based data rather than historical data, include five PROMISE data sets. The hardship of using historical data in quickly changing projects was highlighted, and to solve this problem, AcoForest classifier was developed. The proposed method with 0.685 F-measure yielded better results than Logistic Regression, naive Bayes, decision tree and CoForest.

If a defect prediction is to be performed using static code attributes, data mining methods should be well known. Menzies et al.'s paper published in 2007 is one of the most cited papers regarding how machine-learning methods and performance analyses should be performed (Menzies *et al.*, 2007). This work especially stressed the importance of selected attribute set on prediction performance rather than the predictor. The results obtained performing ten repeats on ten data sets illustrated that naive Bayes classifier was better than J48 with 71% probability of detection. This is the reason naive Bayes was the chosen predictor in our work.

It may not yield good result to apply all learning methods on each data set. For instance Song et al. developed a novel defect prediction framework (Song *et al.*, 2011) and defended this opinion. In addition to this, the differences of their work from Menzies et al.'s work (Fenton and Neil, 1999) were presented. One of them Son's framework seems to be more consistent than Menzies et al.'s framework in terms of results that change according to the selected data sets. Further, it was concluded that the learning scheme should be selected according to the data sets. It may not be the right step to insert defect results in the prediction model. Prior to the prediction, a data cleansing may increase the success of the predictor. In this respect, Kim and Kim's work (Kim *et al.*, 2013) presented a two-phase prediction model. In first phase, deficient reports were eliminated and roughly 70% of the predictions were accurate. This work also encouraged us to perform preprocessing of data.

To measure performance results, Receiver Operator Characteristic (ROC) curves (Fawcett, 2006) and some paramteres obtained from ROC are used for binary classification problems (Wang *et al.*, 2011; Calikli *et al.*, 2009). While performing these operations, our base work is Davis and Goadrich's work (Davis and Goadrich, 2006). The NASA MDP provides a great number of modelling techniques. It is the reason we use NASA MDP data sets while developing the proposed framework. However, limited access of source codes causes constraints for comprehensive research (Ren *et al.*, 2014). At this point, it may be necessary to manually examine defect reports. Herzig et al. (Herzig *et al.*, 2013) investigated the projects and their work showed that 39% of all projects were wrongly labeled as defect-prone. To examine defect reports more accurately, they proposed that projects should be open source. It is called as class imbalance if a great number of defects are on a specific part of the software. Ren et al. proposed a solution (Ren *et al.*, 2014) including two classifiers to cope with this problem, and NASA MDP and SOFTLAP were used for the experimental design. The algorithm AKPCAC produced better results than the others when it was compared in terms of F-measure and Freidmans and Tukey tests.

Area under the curve (AUC) is commonly used for comparing the success of predictors. For instance in Lessman et al.'s work (Lessmann *et al.*, 2008), 20 classifiers were compared with AUC values, and it was concluded that AUC was distinguishing. Linear models especially such as LogReg and LP, produced similar results. They also concluded that if an evaluation were conducted only on the basis of predictor performance, it would be insufficient and that it is necessary to assess the process metrics for evaluation in addition to this, new metrics should be designed.

One of the works that used NASA MDP data sets was Gray et al.s work (Gray *et al.*, 2012). This work focused on data cleansing where some attributes of the metrics obtained from 13 different data sets were removed to make the metrics more suitable for binary classification. If a value was missing, it was replaced with zero. The first results were that data sets should be extended. Thus, it can be discovered whether data points are in general. The second is to determine the repeated data points, using low level metrics. The third is the unknown effect of the repeated data points in classification. Briefly, all of them should be clarified in the future works.

3 Defect Prediction Framework Using Statistical Analysis

3.1 Overview of the Framework

If a defect prediction framework is to be developed, some requirements must be fulfilled. One of them is feature extraction. Therefore, metric values of data sets are prepared prior to the prediction that there are a lot of software to complete this process. However, to compare our work with prior similar works, the same metrics should be generated. Another requirement is that the proposed framework includes learning algorithms depending on the method of the predictor. In addition to these notions, methods and evaluation techniques that increase the success of the predictor should be available.

The proposed framework has three main functions. First, a module that is able to evaluate NASA MDP and SOFTLAB data sets is needed as well as a module of repeated data analysis using statistical methods and a final module that includes data cleansing and performance evaluation. As seen from Fig. 1, feature extraction related to the metrics was completed in the first phase, the obtained metrics are used for repeated data analysis in phase 2, and by producing low level metrics, a performance evaluation is run on the four predictors. These predictors are Bayes.net, naive Bayes, random forest, and J48.

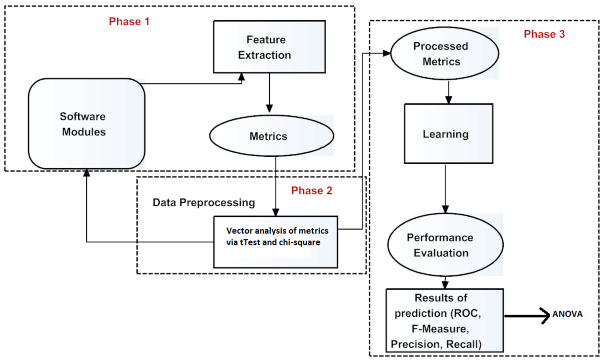
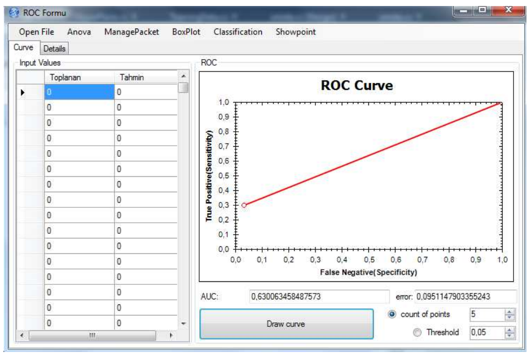


Fig. 1. Outline of the framework



	Toplanan	Tahmin
▸	0	0
	0	0
	0	0
	0	0
	0	0
	0	0
	0	0
	0	0
	0	0
	0	0
	0	0
	0	0
	0	0
	0	0
	0	0
	0	0

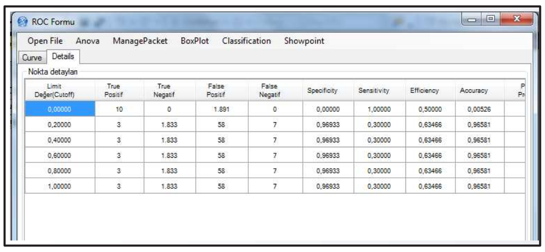
AUC: 0,630063458487573
error: 0,0951147903355243
[● count of points] 5
○ Threshold 0,05

Fig. 2. Default form of the framework

Default form of the framework is seen in Fig. 2. Main screen includes various operations including ROC analysis, statistical tests, feature extraction, and classification. ROC data can be selected through Open File menu from the top left corner of the form. CSV and excel-formatted files are supported. For ROC, user can determine specifications such as count of points and threshold. The frame input values have two columns namely predicted and actual. The ROC curve has various evaluation measures including also precision, sensitivity, and specificity as in Figure 3. Prediction process is conducted via Fig. 4.

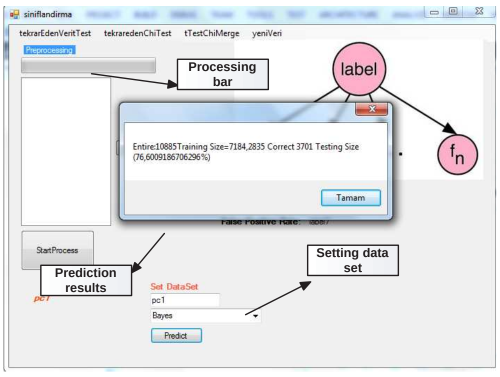
3.2 Statistical Methods

During the experiment, it was necessary to use statistical methods that were compatible with the proposed cleansing algorithm. These methods are ANOVA, t-test and chi-square test. If there is a great number of data that will be examined to discover whether the groups of data show a significant difference, the ANOVA method should be used. Generally, ANOVA is used for comparing the



Limit Değer(Cutoff)	True Positif	True Negatif	False Positif	False Negatif	Specificity	Sensitivity	Efficiency	Accuracy	P Pr
0,00000	10	0	1.891	0	0,00000	1,00000	0,50000	0,00526	
0,20000	3	1.833	58	7	0,96933	0,30000	0,63466	0,96581	
0,40000	3	1.833	58	7	0,96933	0,30000	0,63466	0,96581	
0,60000	3	1.833	58	7	0,96933	0,30000	0,63466	0,96581	
0,80000	3	1.833	58	7	0,96933	0,30000	0,63466	0,96581	
1,00000	3	1.833	58	7	0,96933	0,30000	0,63466	0,96581	

Fig. 3. Roc analysis details



Entire:10885Training Size=7184,2835 Correct 3701 Testing Size (76,6009186706296%)

Fig. 4. Default prediction screen

prediction performances of the learning algorithms in defect prediction works (Song *et al.*, 2011; Lessmann *et al.*, 2008; He *et al.*, 2015). Each square of the groups is calculated as:

$$SS_A = n\sum(\overline{Y}_{.j} - \overline{Y}_{..})^2 \tag{1}$$

The mean of a particular group is represented by $\overline{Y_{.j}}$. Here $\overline{Y_{..}}$refers to the means of the full data. The count of the observations is represented by n. Sum of squares of the groups is formulated as:

$$SS_T = SS_A + SS_{\frac{S}{A}} \tag{2}$$

While calculating the sum of squares, some errors may be observed. These errors (called as variations) can be defined as:

$$SS_{\frac{S}{A}} = \sum\sum(\overline{Y}_{ij} - \overline{Y}_{.j})^2 \tag{3}$$

$$df_A = a - 1, df_{\frac{S}{A}} = a(n-1) = N - a \tag{4}$$

As shown in the Eq. 4, each sum of squares has a degrees of freedom that is unique. Here, number of groups is a. The variable n refers to the number

Table 1. ANOVA

SS_A	df_A	MS_A	F
$SS_{\frac{S}{A}}$	$df_{\frac{S}{A}}$	$MS_{\frac{S}{A}}$	
SS_T			

of observations that are computed in each group. The F score is obtained by dividing the variance of between groups by the variance of within groups, as in the Eq. 5.

$$F = \frac{MS_A}{MS_{\frac{S}{A}}} \tag{5}$$

The table of ANOVA was arranged as in Table 1 If ANOVA is applied on ony two groups of the data sets, it becomes t-test. Details of t-test are:

a_{11}= mean of sample 1, a_{22}= mean of sample 1,
n_1= number of subjects in sample 1,
n_2= number of subjects in sample 2 where

$$s_1^2 = (\sum (a_1 - a_{11})^2 / n_1) \tag{6}$$

is the variance of sample 1 (6), and

$$s_2^2 = (\sum (a_2 - a_{22})^2 / n_2 \tag{7}$$

is the variance of sample 1 (7), where

$$t = \frac{\overline{x}1 - \overline{x}2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2} - 2r[\frac{s_1}{\sqrt{n_1}}][\frac{s_2}{\sqrt{n_2}}]}} \tag{8}$$

The chi-square test is simply used for the determination of the association between two categorical variables. In defect prediction, defect-prone and not defect-prone are our categories. To specify whether there is an association between our categories, we used the formula as follows:

$$x^2 = \sum \frac{(valueObserved - valueExpected)}{valueExpected} \tag{9}$$

While deciding whether our hypothesis is acceptable, we used a probability level of 0.05 as our critical value.

3.3 Feature Extraction

Feature extraction is obtaining properties of software modules on the basis of certain standards (Halstead, 1977; McCabe, 1976). The most used ones are LOC, McCabe, and Halstead. Also, our experimental data sets were retrieved from NASA MDP and SOFTLAB generated metrics using these standards. In our proposed framework, the metrics of projects can be extracted regardless of programming languages, such as C, C++, Java, and C#. However, we have used NASA MDP and SOFTLAB data sets rather than the untested metrics of arbitrary selected software to compare our work with Gray's work.

3.4 Data Removal

While predicting defects the quality of data must be at a specific level. If the quality level is not at least at the desired level, various problems occur. These problems affect the success of the classification and prediction results. Class Imbalance, unsuitable data for classification and same data sets in metrics are considered some of the problems. For instance, the KC4 data of NASA MDP have metrics that have similar meaning to row count, such as number of lines and loc total. If one of these values is not removed from the metrics, the results of classification will be incorrectly obtained for a specific ratio. A metric representing either defect-prone or not is available in NASA MDP and SOFTLAB. This metric is called as "defects" and has been made more suitable for binary classification by assigning false and true values. We have two hypotheses to describe defect data sets:H_0: If p is greater than 0.05, then data sets are not different, H_a: If p is not less than 0.05, then data sets are different.

We can explain data preprocessing as two phases. In the first phase, statistical operation is conducted on data. These operations include vectorial calculation of all the data samples with the help of some statistical tests such as t-test and chi-square.

One of the test methods for data sets was the t-test. While using this test, it is checked that whether means of our hypothesis are different from the actual means. If a significant difference is discovered, operations of prediction have been carried out. Otherwise, samples have been removed prior to the usage of data samples in the prediction model.

Our last test is the chi-square. In this test, we attempted to discover whether the two groups are different from each other. If the obtained p-value is less than 0.05, it is figured out that the two groups are not different; therefore, they can be utilised for prediction. Public availability of defect data sets and detailed examination of defect comments provide more accurate results regarding data sets. In addition to this, data sets are not open source, which causes various obstacles in situations that requires manual examination. Gray's thesis stressed that using low level metrics facilitates to differentiating modules

Table 2. Details of NASA MDP and Softlab data sets

Name	Language	Attributes	Instances	Defective Instances%	Missing Values
ar1	C	29	241	8	-
ar3	C	29	125	12	-
ar4	C	29	213	18	-
ar5	C	29	71	21	-
ar6	C	29	201	14	-
cm1	C	40	327	12	-
jm1	C	21	10878	19	25
kc1	C++	21	2107	15	-
kc2	C++	21	521	20	-
kc3	Java	40	458	9	-
pc1	C	40	1107	7	-
pc2	C	40	5589	0.4	-
pc3	C	40	1563	10	-
pc4	C	40	1458	12	-
pc5	C++	39	17186	3	-

(Gray, 2013). However, this process should be done very carefully because reduction of input space may adversely affect the classification.

Our developed preprocessing algorithm includes deriving the character count. The metric table has been extended in this way. On the manual examination of 15 data sets, we have detected the relationship that character count is roughly 30 times of that of the value loCode. We have unveiled and implemented the formulation on all data sets as follows:

$$\begin{aligned} cCount &= lCode * 30 \\ cS &= totalnumber\_of\_operations \end{aligned}$$

Our data sets have metrics including missing values. We have assigned zero to missing values instead of reducing them (Gray, 2013). The proposed data cleansing algorithm was carried out in three phases. In the first phase chi-square and t-test are applied to all data sets. These tests are needed due entirely to the verification of repeated data points that are consisted of numbers suitable for a statistical operation. This operation facilitates distinguishing the effects of metric values. Class size (cS) is also a low level metric but it is not available in our experimental data sets before preprocessing. We added character counts and cS to the data, which passed through the first phase successfully, the extension of metrics was implemented, and finally missing values were filled with zero afterwards. The algorithm is seen in Algorithm 1. Initially, required statistical test are applied on data sets to determine the weights of the metrics. In the Step 8, all the instances are converted to the vectorial format by giving weighting to the each metrics. Step 9 is for the calculation of the euclidean distances of the instances which is measured by using Equation 11 where e is euclidean distance and $p_{i/j}(x)$ is the distribution function (Helén and Virtanen, 2007).

$$e(x)_{i..j} = \int_{-\infty}^{\infty} [p_1(x) - p_2(x)]^2 dx \tag{10}$$

Algorithm 1 Cleansing of Repeated Data

```
procedure CLEAN($Data$)                              ▷ The Data:NASA, Softlab
    N:ar1,ar2,ar3,ar4,ar5,ar6,cm1,jm1,kc1,kc2,kc3
    $X_i$ : Dataset selected from N
    while $i \neq 15$ do
        $x_i \leftarrow N$
        $Chi - square(x_i)$
        $T - Test(x_i)$
        Binary Combination $(v_i \leftarrow x_i)$
        $e(v_i)_{i..j} = \int_{-\infty}^{\infty} [p_i(x) - p_j(x)]^2 dx$
        $v_i + = Lmetrics(x_i)$
        $M = Count(x_i)$                             ▷ Calculate count of instance
        for j=0 to M do
            if $x_{ij} ==' Missing'$ then
                $x_{ij} = 0$
            end if
        end for
    end while
    return $Data$
end procedure
```

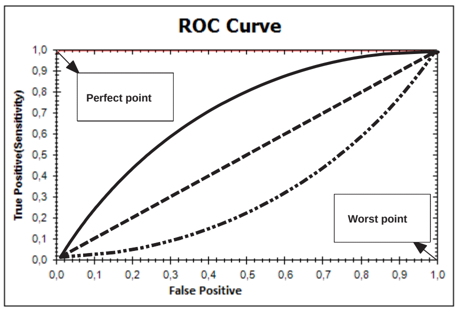


Fig. 5. Details of the ROC curve

4 Evaluation of the Framework

4.1 Data Sets

Our data sets, as described in the prior sections, are comprised of NASA MDP and SOFTLAB repositories. The count of attributes of the data sets changed between 21 and 40. Even though NASA MDP data sets vary according to the type of the project, McCabe and Halstead-based metric tables are similar. While using data sets we have observed the same case in the data retrieved from SOFTLAB. This incident complicates the conducting of experimental design. If the data of projects that were prepared for prediction of defects were accepted as a common standard, works related to prediction of defects would speed up. Sample and attribute count of the data sets are shown in Table 2. Before the use of statistical tests, all the data sets were exposed

Table 3. Confusion matrix

		PREDICTED	
		nfp	**fp**
REAL	**nfp**	TN	FP
	fp	FN	TP

to normalization that normalizes all numeric values in the data sets. This process was conducted by using smallest and the largest value of the data sets depending on the scale and transition parameters. Defect prediction data sets can be selected through writing its name as shown in Fig. 4.

4.2 Evaluation Parameters

Since prediction of defects is on the basis of binary classification, if any evaluation of performance is conducted, initially the confusion matrix should be built (Weyuker *et al.*, 2008; Caglayan *et al.*, 2015). While predicting, some definitions are created such as fault prone(fp) and not fault prone (nfp). If any module of software does not include defect, and the bias is correct, the module is labeled as TN. If any module of software includes defects that are wrongly biased, the module is labeled as FN, otherwise modules that do not include any defect and wrongly biased, are labeled as FP. If the module includes a defect and is correctly biased, then it is labeled as TP. After these definitions Table 3 was generated.

Precision =TP/TP + FP;
Recall =TP/TP + FN; TPR = (TP/TP + FN)
FPR = (FP/FP + TN)

Details of ROC that measures the effectiveness of the classification are seen in Fig. 5. The y-axis records recall calculated by the formula TP/TP+FN. In the same way the value expressed in the x axis is calculated by the formula TP/TP+FN. It is understood that values used for the search effectiveness, including recall and FPR, are also used in ROC. The area under the curve is AUC. The best point of the curve is (1,0) and that means there is no wrong prediction. The worst point of the curve is (0,1) and that means there is no right prediction. The area that is above the diagonal is generally the desired result. The other area is under the diagonal means the success of classification is not at sufficient level. There is an inverse relationship between precision and recall and the general result is the F-measure, calculation of these two values using harmonic means a value is obtained as in Fig. 6. As the F-measure becomes closer to one better performance is acquired.

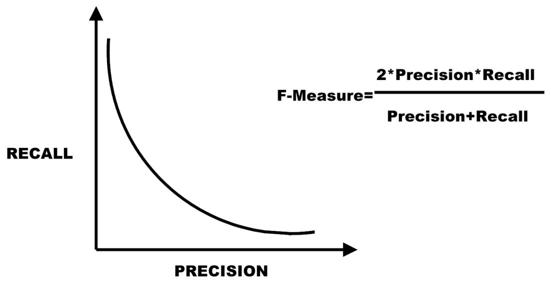


Fig. 6. Correlation between Precision and Recall

Table 4. Changes of precision of all data sets

Name	Bayes%	naive Bayes%	J48%	RandomForest%
ar1	-3	+2	+7	-12
ar3	+15	+5	0	0
ar4	+8	+17	+9	+9
ar5	0	0	0	0
ar6	0	0	+3	-3
cm1	-9	+6	+2	-1
jm1	-3	0	+2	-6
kc1	0	0	0	0
kc2	-4	0	+5	+1
kc3	-8	+3	+12	+4
pc1	+22	+3	-8	-1
pc2	+2	-4	+14	+4
pc3	-3	+24	-1	-11
pc4	-2	-5	0	-8
pc5	-1	0	0	0

4.3 Experimental Design

4.3.1 Experimental Conditions

We have investigated on 15 data sets, as seen in Table 2. These data were prepared with the compliance of Weka used for data mining operations as the format arff. Before the experimental design, preprocessing data sets were run on the four different classifiers including Bayes, naiveBayes, J48 and randomforest, consequently the results obtained from Weka were saved for the comparison of with our algorithm. It is hard to investigate data sets as arff due to the file format, so we have converted file formats to txt. Thus, using parsing methods on the data sets, all operations specified in preprocessing have been correctly conducted. Prior to the preprocessing predictor, performances of the classifiers are seen in Table 7.

In binary classification problems, data sets are generally divided into 66% training and 34% testing sets, or 10 fold cross-validation is used. Prior to the preprocessing, selected algorithms yielded results using 66% training and 34% testing sets. This allocation changes depending on the size of the data sets and the selection of the correct ratio is important in terms of prediction performance. Performance of our proposed preprocessing algorithm has been measured with developed framework coded with C# that includes the Weka library. The graphs of the analyses have been drawn with ZedGraph. Character counts of all data sets were calculated; afterward all of them were added to the metrics adding an attribute with the label ”cCount”.

4.4 Results

This section explores the ability of the classifiers to predict defects when applied not only on large data sets but also on data sets with few samples. After applying the preprocessing algorithm to data sets presented in Fig. 4, the performances of the predictors, including Bayes, naive Bayes, random forest and J48 have been observed. As seen from Table 4, naive Bayes yielded a precision enhancement of 0.24 on the data set of pc3. Due to the 40 attributes of pc3 from 1563 samples, we obtained better results than the first situation. Likewise, although jm1 data set includes 10878 samples, it did not have increased success of predictors due to an insufficient number of attributes (for instance 20). This reveals the importance of attributes in the case of performance, while predicting defects.

As the samples of data sets used in the experimental design reduce, the reliability of the results of the analysis decreases. For instance, one of the data sets is ar3 which has 63 samples. After preprocessing, this data set did not produce a reliable result (100%) due to lack samples, as seen from Table 4.

Values of precision are shown from Fig. 7. In all these figures, the variation in the lines represents the changes of precision value across the data sets. The precision scores of the data sets prior to the preprocessing and after the preprocessing are recorded by a straight line and a dashed line respectively. In such graphs, it would be desirable to overlap two lines. It is clear that naive Bayes has yielded the best result in terms of overlapping. Bayes classifier increased success of predictors on pc1, pc2, ar3, and but did not demonstrate consistency in general. The best improvement on data sets resulted from using random forest with 4% pc2 after preprocessing. Even though J48 had different situations on all data sets, note that this algorithm features the minimum breaking points when compared with others. Table 7 and Table 6 show the AUC and error values of all data sets on four predictors before and after preprocessing respectively. The overall AUC has been increased 3.91% on average.

In addition, all data sets were analyzed to understand how correlated metric values might be. This analysis Spearman's correlation measures the strength of linear relationship of desired pairwise value. Correlation coefficient rs changes between -1 and 1. If obtained result is close to ± 1, this means there is a strong monotonic relationship. Spearman correlation is a preferable evaluation parameter for defect prediction works (Caglayan *et al.*, 2015). Complexity parameters cyclomatic_complexity and v(g) were selected to investigate a pairwise analysis with cS to find out how effective low level metrics on the defect-proneness of data sets. The results presented in Fig. 9-10 show that the correlation between cS and complexity is monotonically increasing.

Table 5. Split rates of the samples and F-Scores

Name	Samples	Count of values	Set of training	Set of Testing	F-score
ar1	121	7471	79	42	0.873
ar3	63	3596	41	22	0.939
ar4	107	6603	70	37	0.757
ar5	36	2201	23	13	0.752
ar6	101	6231	66	35	0.766
cm1	327	12426	215	112	0.812
jm1	10885	500710	7184	3701	0.754
kc1	2109	96991	1391	718	0.798
kc2	522	23989	344	178	0.817
kc3	458	37515	302	156	0.845
pc1	1109	51014	731	378	0.754
pc2	5589	424726	3688	1901	0.978
pc3	1563	59394	1031	532	0.787
pc4	1458	113685	962	496	0.864
pc5	17186	1374840	11342	5844	0.963

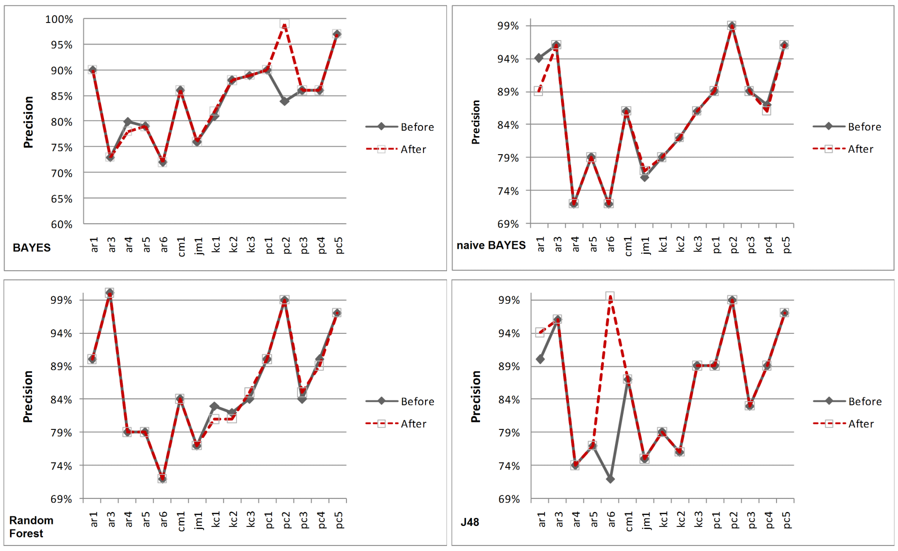


Fig. 7. Comparison of all data sets for predictors

Rs values of the experiment change between 0.5-0.9. Obtained results are a good indicator of the significance of class design in the density of defects.

5 Discussion

While performing the investigation, one of the discussion points was whether repeated data of NASA MDP were in general. Results of the analysis show that all data have roughly 66% repeated data points. However, it may not be sufficient to say that there is a general bias on all data sets of NASA MDP and SOFTLAB. It is difficult to validate our method so that NASA data sets

Table 6. Results of AUC of four predictors after preprocessing

Data set	**Bayes**	**naive Bayes**	**RandomForest**	**J48**
Ar6	0,5 ± 0,131	0,563 ± 0,133	0,563 ± 0,133	0,58 ± 0,13
Cm1	0,514 ± 0,09	0,279 ± 0,09	0,27 ± 0,094	0,09 ± 0,06
Jm1	0,764 ± 0,01	0,569 ± 0,01	0,569 ± 0,01	0,58 ± 0,012
Kc1	0,571 ± 0,02	0,61 ± 0,02	0,61 ± 0,02	0,59 ± 0,02
Kc2	0,715 ± 0,051	0,65 ± 0,05	0,65 ± 0,05	0,71 ± 0,05
Kc3	0,64 ± 0,08	0,65 ± 0,08	0,65 ± 0,08	0,63 ± 0,08
Pc1	0,54 ± 0,012	0,56 ± 0,01	0,56 ± 0,01	0,58 ± 0,012
Pc2	0,66 ± 0,094	0,63 ± 0,09	0,63 ± 0,09	0,5 ± 0,09
Pc3	0,59 ± 0,0376	0,67 ± 0,03	0,67 ± 0,03	0,56 ± 0,03
Pc4	0,64 ± 0,04	0,68 ± 0,03	0,68 ± 0,03	0,72 ± 0,03
Pc5	0,86 ± 0,01	0,71 ± 0,02	0,71 ± 0,02	0,65 ± 0,02

Table 7. Results of AUC of four predictors before preprocessing

Data set	**Bayes**	**naive Bayes**	**RandomForest**	**J48**
Ar6	0,5 ± 0,131	0,58 ± 0,133	0,58 ± 0,133	0,58 ± 0,13
Cm1	0,542 ± 0,075	0,58 ± 0,07	0,58 ± 0,07	0,54 ± 0,07
Jm1	0,44 ± 0,013	0,18 ± 0,011	0,18 ± 0,011	0,16 ± 0,01
Kc1	0,582 ± 0,002	0,60 ± 0,02	0,60 ± 0,02	0,59 ± 0,02
Kc2	0,727 ± 0,05	0,65 ± 0,05	0,65 ± 0,05	0,71 ± 0,05
Kc3	0,15 ± 0,07	0,35 ± 0,09	0,35 ± 0,09	0,28 ± 0,08
Pc1	0,722 ± 0,05	0,46 ± 0,06	0,46 ± 0,06	0,29 ± 0,06
Pc2	0,75 ± 0,08	0,31 ± 0,13	0,31 ± 0,13	0-0
Pc3	0,732 ± 0,03	0,53 ± 0,04	0,53 ± 0,04	0,2 ± 0,03
Pc4	0,647 ± 0,04	0,68 ± 0,03	0,68 ± 0,03	0,72 ± 0,03
Pc5	0,86 ± 0,01	0,73 ± 0,02	0,73 ± 0,02	0,65 ± 0,02

are not representative for real world situations. We addressed this problem by using SOFTLAB data sets that its data are from communication industry. In this respect, our proposed method should attempt to validate on open source of projects. To compare our work with Grays wok, data sets of other open source modules were not involved to the experimental design.

Even though researches published in last five years are focused on process metrics that yielded promising results (Rahman and Devanbu, 2013; Wiese *et al.*, 2014), code metrics have some gaps that are worthy to explore (Oliveira *et al.*, 2014; Zhang *et al.*, 2014). One of them is the quality of defect data sets. This quality is associated with the way of data collection. As the defect data sets are generally prepared by combining all related developer's comments, they may have missing or noisy data points. In order to cope with this problem, the data are re-sampled or reduced by using particular preprocessing techniques. SMOTE is one of the widely used sampling strategies for defect prediction (Pears *et al.*, 2014). However it is sensible to combine a sample reduction method with an over-sampling technique (Chen *et al.*, 2015). In this respect, our method is distinctive that combines metric derivation with cleansing of repeated data. Table 5 which indicates details of the experiment, has similar F-scores yielded from the classification. F-scores are the mean of weighted averages of the algorithms including Bayes, naive Bayes, Random-

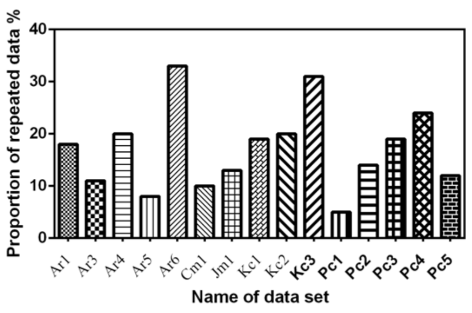


Fig. 8. Proportion of repeated data in experimental data sets

Table 8. One-way ANOVA results of the predictors.

ANOVA	**SS**	**DF**	**F**	**P**
Between	0,198033	3	3,432675	0,022828
Within	0,360567	25		

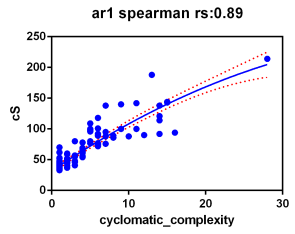
Forest, and J48. This evaluation parameter is the harmonic mean of precision and recall. These tests appear to be adequate to determine the difference of samples both between each other and outside of groups. However, if the range of attributes of the data sets changes, a different statistical method may be required to trying.

ANOVA results of the experimental predictors are presented in Table 8. Looking these results $((p = 0.02) < 0.05)$, we have noticed that all predictors Bayes, naive Bayes, random forest, and J48 yielded different results.

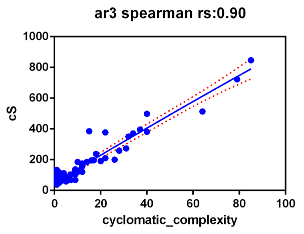
6 Conclusion and Future Works

Regardless of the attributes of the data sets and their generation method, they have repeated data points at a certain ratio. To detect these repeated data, data sets should be exposed to a preprocessing prior to the prediction. Inherently performed methods for defect prediction should be completed regarding essential properties of binary classification. Removal of unsuitable data for binary classification may be a solution. However, every new data to be added to data sets creates an adverse impact that leads to overfitting. This reduces the success of learning.

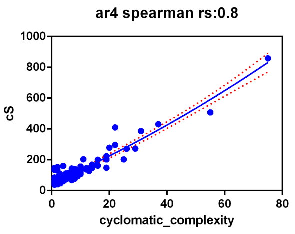
It still remains unclear that whether data sets of other open source projects have similar repeated data points, as in NASA MDP. We have detected in the experimental design that all data set of NASA MDP and SOFTLAB have repeated data points in a specific ratio. This ratio changes depending on the type of the data set but was roughly detected as 17% on average as in Fig. 8.



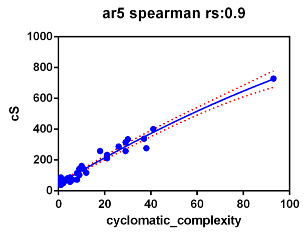
(a) ar1



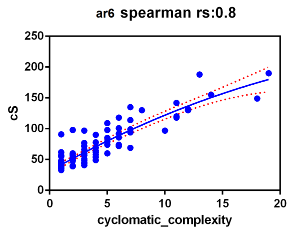
(b) ar3



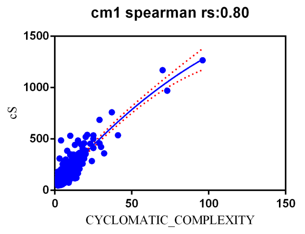
(c) ar4



(d) ar5



(e) ar6



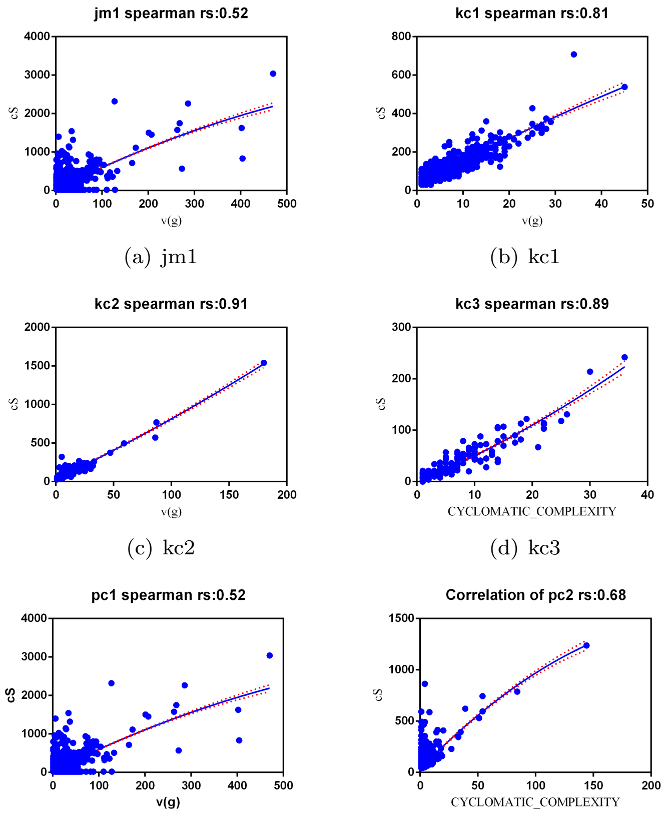
(f) cm1

Fig. 9. Spearman correlations of data sets I

The use of low level metrics facilitates the detection of more data points than the approach that uses traditional metrics. Therefore, the differential capability of the used metrics has been increased. If defect data sets are not derived from open source of projects, extending software metrics becomes difficult. At this point, new metrics can be derived by investigating relationships of old metrics. We have added cS and cCout to the metrics using the LOC total through the formula stated as in Section 3.3. Thus, we have unveiled more repeated data points than after the preprocessing.

Using ANOVA, chi-square and t-test on data sets that were compared with both in groups and out of groups, the extension of metrics was completed with data that passed from these tests. Values of data sets having under a p-value of 0.05 have been approved to our hypothesis.

The values of AUC of the proposed method with post preprocessing values on four classifiers are shown from Table 6. These results indicate that the cleansing-based approach outperforms the one without preprocessing. It is



(a) jm1 (b) kc1

(c) kc2 (d) kc3

(e) pc1 (f) pc2

Fig. 10. Spearman correlations of data sets II

important to note that AUC figures of Bayes, random forest and J48 are similar. However, the success of prediction has been increased in the pc3 data set using naive Bayes with the ratio of 24%. This ratio is high when compared with other predictors.

The success of the preprocessing affects classifying. Differentiating and comprehending software modules more clearly provides precise training and testing data sets. Future works may include new methods to derive of low level metrics, and the proposed methods should be validated on open source systems with manual examination.

References

Bell, R. M., Ostrand, T. J., and Weyuker, E. J. (2013). The limited impact of individual developer data on software defect prediction. *Empirical Software*

Engineering, **18**(3), 478–505.

Caglayan, B., Turhan, B., Bener, A., Habayeb, M., Miransky, A., and Cialini, E. (2015). Merits of organizational metrics in defect prediction: an industrial replication. In *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*, volume 2, pages 89–98. IEEE.

Calikli, G., Tosun, A., Bener, A., and Celik, M. (2009). The effect of granularity level on software defect prediction. In *Computer and Information Sciences, 2009. ISCIS 2009. 24th International Symposium on*, pages 531–536. IEEE.

Chen, L., Fang, B., Shang, Z., and Tang, Y. (2015). Negative samples reduction in cross-company software defects prediction. *Information and Software Technology*, **62**, 67–77.

Czibula, G., Marian, Z., and Czibula, I. G. (2014). Software defect prediction using relational association rule mining. *Information Sciences*, **264**, 260–278.

Davis, J. and Goadrich, M. (2006). The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240. ACM.

Elish, K. O. and Elish, M. O. (2008). Predicting defect-prone software modules using support vector machines. *Journal of Systems and Software*, **81**(5), 649–660.

Fawcett, T. (2006). An introduction to roc analysis. *Pattern recognition letters*, **27**(8), 861–874.

Fenton, N. E. and Neil, M. (1999). A critique of software defect prediction models. *Software Engineering, IEEE Transactions on*, **25**(5), 675–689.

Gray, D., Bowes, D., Davey, N., Sun, Y., and Christianson, B. (2012). Reflections on the nasa mdp data sets. *Software, IET*, **6**(6), 549–558.

Gray, D. P. H. (2013). Software defect prediction using static code metrics: formulating a methodology.

Hall, T., Beecham, S., Bowes, D., Gray, D., and Counsell, S. (2012). A systematic literature review on fault prediction performance in software engineering. *Software Engineering, IEEE Transactions on*, **38**(6), 1276–1304.

Halstead, M. H. (1977). *Elements of software science*, volume 7. Elsevier New York.

He, P., Li, B., Liu, X., Chen, J., and Ma, Y. (2015). An empirical study on software defect prediction with a simplified metric set. *Information and Software Technology*, **59**, 170–190.

Helén, M. and Virtanen, T. (2007). Query by example of audio signals using euclidean distance between gaussian mixture models. In *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*, volume 1, pages I–225. IEEE.

Herzig, K., Just, S., and Zeller, A. (2013). It's not a bug, it's a feature: how misclassification impacts bug prediction. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 392–401. IEEE Press.

Keivanloo, I., Forbes, C., Hmood, A., Erfani, M., Neal, C., Peristerakis, G., and Rilling, J. (2012). A linked data platform for mining software repositories. In

Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on, pages 32–35. IEEE.

Kim, D., Tao, Y., Kim, S., and Zeller, A. (2013). Where should we fix this bug? a two-phase recommendation model. *Software Engineering, IEEE Transactions on*, **39**(11), 1597–1610.

Kim, S., Zhang, H., Wu, R., and Gong, L. (2011). Dealing with noise in defect prediction. In *Software Engineering (ICSE), 2011 33rd International Conference on*, pages 481–490. IEEE.

Lessmann, S., Baesens, B., Mues, C., and Pietsch, S. (2008). Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *Software Engineering, IEEE Transactions on*, **34**(4), 485–496.

Li, M., Zhang, H., Wu, R., and Zhou, Z.-H. (2012). Sample-based software defect prediction with active and semi-supervised learning. *Automated Software Engineering*, **19**(2), 201–230.

Liebchen, G. A. and Shepperd, M. (2008). Data sets and data quality in software engineering. In *Proceedings of the 4th international workshop on Predictor models in software engineering*, pages 39–44. ACM.

Linstead, E., Bajracharya, S., Ngo, T., Rigor, P., Lopes, C., and Baldi, P. (2009). Sourcerer: mining and searching internet-scale software repositories. *Data Mining and Knowledge Discovery*, **18**(2), 300–336.

Madeyski, L. and Jureczko, M. (2015). Which process metrics can significantly improve defect prediction models? an empirical study. *Software Quality Journal*, **23**(3), 393–422.

McCabe, T. J. (1976). A complexity measure. *Software Engineering, IEEE Transactions on*, (4), 308–320.

Menzies, T., K. R. P. D. (2015). The promise repository of empirical software engineering data.

Menzies, T. and Koru, G. (2013). Predictive models in software engineering. *Empirical Software Engineering*, **18**(3), 433.

Menzies, T., Greenwald, J., and Frank, A. (2007). Data mining static code attributes to learn defect predictors. *Software Engineering, IEEE Transactions on*, **33**(1), 2–13.

Oliveira, P., Valente, M. T., and Paim Lima, F. (2014). Extracting relative thresholds for source code metrics. In *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on*, pages 254–263. IEEE.

Pears, R., Finlay, J., and Connor, A. M. (2014). Synthetic minority oversampling technique (smote) for predicting software build outcomes. *arXiv preprint arXiv:1407.2330*.

Rahman, F. and Devanbu, P. (2013). How, and why, process metrics are better. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 432–441. IEEE Press.

Rajesh Vasa, M. L. and Jones, A. (2010). Helix - Software Evolution Data Set.

Ren, J., Qin, K., Ma, Y., and Luo, G. (2014). On software defect prediction using machine learning. *Journal of Applied Mathematics*, **2014**.

Shepperd, M., Song, Q., Sun, Z., and Mair, C. (2013). Data quality: Some comments on the nasa software defect datasets. *Software Engineering, IEEE Transactions on*, **39**(9), 1208–1215.

Song, Q., Jia, Z., Shepperd, M., Ying, S., and Liu, S. Y. J. (2011). A general software defect-proneness prediction framework. *Software Engineering, IEEE Transactions on*, **37**(3), 356–370.

Succi, G., Pedrycz, W., Stefanovic, M., and Miller, J. (2003). Practical assessment of the models for identification of defect-prone classes in object-oriented commercial systems using design metrics. *Journal of Systems and Software*, **65**(1), 1–12.

Verma, R. and Gupta, A. (2012). Software defect prediction using two level data pre-processing. In *Recent Advances in Computing and Software Systems (RACSS), 2012 International Conference on*, pages 311–317. IEEE.

Wang, H., Khoshgoftaar, T. M., and Seliya, N. (2011). How many software metrics should be selected for defect prediction? In *Twenty-Fourth International FLAIRS Conference*.

Wang, S. and Yao, X. (2013). Using class imbalance learning for software defect prediction. *Reliability, IEEE Transactions on*, **62**(2), 434–443.

Weyuker, E. J., Ostrand, T. J., and Bell, R. M. (2008). Do too many cooks spoil the broth? using the number of developers to enhance defect prediction models. *Empirical Software Engineering*, **13**(5), 539–559.

Wiese, I. S., Côgo, F. R., Ré, R., Steinmacher, I., and Gerosa, M. A. (2014). Social metrics included in prediction models on software engineering: a mapping study. In *Proceedings of the 10th International Conference on Predictive Models in Software Engineering*, pages 72–81. ACM.

Zhang, F., Mockus, A., Keivanloo, I., and Zou, Y. (2014). Towards building a universal defect prediction model. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 182–191. ACM.

Zimmermann, T., Nagappan, N., Gall, H., Giger, E., and Murphy, B. (2009). Cross-project defect prediction: a large scale experiment on data vs. domain vs. process. In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 91–100. ACM.