An empirical analysis of threshold techniques for identifying faulty classes in object oriented software systems

Navneet Kaur
and
Hardeep Singh
Guru Nanak Dev University, Amritsar, India

The fault-proneness prediction is one of the extensively researched areas in the Software Engineering domain to ameliorate the software quality and to lowers the testing expenses. The existing literature has proved the presence of large amount of work attained the statistical validation of software metrics by utilizing them in the development of fault prediction models, where, both statistical and machine learning techniques were engaged into the construction of the models being capable of identifying faulty and non-faulty classes. On the contrary, the research area involving the investigation of threshold concept has not gained sufficient maturity. An effective threshold technique can assist in the identification of optimal cut-off values of the software metrics which can discriminate the faulty from non-faulty classes with minimal misclassification rate. The idea of threshold calculation can make the application of the existing metrics in software industries, a much easier task. As the developers only need to know the cut-off values which can help them to concentrate on the specific classes that exceeds the computed thresholds. Also, the presence of peculiarity in the software metric index can alert the testers and in turn helps them to disburse the resources systematically. The current study empirically validated and compared the discriminating strength of two threshold techniques, i.e., ROC curve and Alves Rankings, on the publicly available dataset. This study selected twenty object oriented measures for the process of threshold calculation. Furthermore, Wilcoxon signed rank test was used to enquire the classification difference between the aforementioned threshold techniques. The outcome from the statistical analysis revealed the better predictive capability of ROC curve than the Alves Rankings.

Keywords: Fault-proneness, Threshold, ROC curve, Alves Rankings, Object Oriented metrics.

1. INTRODUCTION

In Software Engineering domain, the main objective of the software developers is to produce high quality software system. The prerequisite for improving the software quality is the existence of some measures which are proficient in evaluating the quality in early phases of the software development process. A substantial amount of research efforts has been devoted in formulating the software measures, capable of quantifying different quality attributes Rosenberg and Hyatt [1997], Rajaraman and Lyu [1992]. The usability of the proposed measures in an industrial environment is merely dependant on their practical validation. As a result, the researchers demonstrated their practical validation by utilizing them in the prediction of various quality attributes such as fault-proneness, maintainability, and change-proneness Aggarwal et al. [2007], Van Koten and Gray [2006], Khoshgoftaar et al. [2001], Ostrand et al. [2005]. Despite this, the software measures are rarely exercised in the industrial environment due to unavailability of the proper metric thresholds. The identification of an optimal cut-off value of software measure is essential in Software Engineering to produce necessary guidelines for the process of decision-making Alves et al. [2010]. Over the last two decades, a plethora of prediction models, backed by statistical and machine learning approaches, has been developed to identify the faulty classes Mahajan et al. [2015], Thwin and Quah [2002], Kaur and Malhotra [2008]. The fault prediction models can enhance the quality of the software system by locating the faults before the delivery of system to the customer. But, their implementation in the software industries is quite cumbersome, as it involves

the construction and execution of models on a regular basis Shatnawi et al. [2010]. On the other hand, the application of thresholds in the software industries is quite easy. The identification of thresholds in early stages of software development process can help the developers to focus on the classes in which the metric values exceeded the computed thresholds. Similarly, in testing phase, the threshold concept can help in the systematic distribution of the testing resources, such as, time and efforts. As, there is no need to examine all of the software classes, the testers can put efforts only on classes having metric values greater than the identified thresholds. But, despite of their advantage over prediction models, work done in the field of threshold area is rather low. The current study aims to probe the usefulness of threshold techniques in the discipline of fault prediction by assessing their proficiency in separating the faulty from non-faulty classes. In order to do that, we calculated the thresholds of Object Oriented (OO) measures by employing ROC curve and Alves Rankings statistics. We also compared the predictive performance of the aforementioned techniques to find out the one with efficient prognostic performance. It is pertinent to mention here that a study accomplishing the similar motive is present in the literature Boucher and Badri [2017]. However, there is a call for the re-examination of this subject for two reasons. First, the existing study examined the statistics proficiency in giving the appropriate thresholds of only Chidamber and Kemerer (CK) metric suite, whereas, the thresholds of other widely used OO measures also needs to be tested for their relationship with fault-proneness. Second, it is necessary to assess the applicability of the findings of the study Boucher and Badri [2017] on other software systems also, the present study has performed the experiment on varied dataset. Based on the analysis of the studies related to the similar subject, a series of steps are limned in Figure 1 and the current study has been compiled accordingly. The public dataset was used to assess the applicability of the threshold concept on the software measures. Furthermore, Univariate Logistic Regression (URL) was applied to confirm the predictive capability of the considered software measures toward fault-proneness. Only those measures which exhibited essential relationship with fault-proneness were selected for the threshold calculation process. At last, the predictive performances of metric thresholds computed through the ROC curve and Alves Rankings were compared with the Wilcoxon signed rank test. The selected test is non-parametric test and we adopted this method instead of the parametric as the application of the latter one can give unreliable results. In case of parametric test, the characteristics of the dataset must hold three mandatory conditions, i.e., independency, homoscedasticity, and normality (add citation) and the fault dataset opposed these conditions.

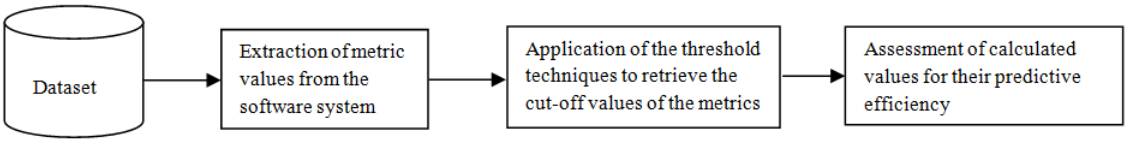

Figure 1. Steps followed for threshold calculation in the current study

The current study also assessed the cross-version threshold applicability by employing the values computed from one version, on other versions of the same software system. This concept of generalization of threshold values among different versions of the same software seems reliable, but in case of different systems this idea does not seem to give authentic results. As the threshold proved good on one software system might not work on other systems, due to possession of distinct characteristics. The software systems might be written in different languages or belong to different organizations. In such cases, a criteria need to be defined, based on which we can put the software systems in a similar category and generalize the threshold values retrieved from one system to others. Several criteria were mentioned in the previous studies for the selection of

software systems in the experiment of cross-project fault prediction and the most common one is the selection of systems based on their similar external characteristics. In this study, we define two conditions for the selection of systems in the cross-project experimentation.
The contribution of the current study is threefold.

1) Empirically examines the existence of threshold values of software metrics that can efficiently group the faulty and non-faulty classes.
2) Examines the capability of ROC curve and Alves Rankings in determining the optimal threshold values.
3) Compares the predictive performances of the above threshold techniques.

The rest of the paper is organized as follows. Section 2 contains the overview of studies investigated the relationship between software metric thresholds and fault-proneness. Section 3 comprises the description about the considered datasets, software metrics, threshold techniques, performance evaluation measures, and comparison method that were opted to execute the whole experiment. Section 4 presents the detailed experimental results. Section 5 takes up various threats to the current study. And in the last section, we present the conclusions and some recommendations about the future work.

2. RELATED WORK

Several studies have been conducted to probe the effectiveness of software measures for the development of prediction models identifying the faulty classes. In this section, we discussed the prior work conducted to identify the relationship between metric thresholds and fault-proneness. The existing literature indicates the presence of a small amount of work incorporating the determination of threshold values based on the researcher experience or by using the statistical techniques. McCabe [1976], based on his experience, declared the fixed threshold value of the complexity metric . According to the author, if the value of complexity metric exceeds 10, then the modularization of the sub-functions is crucial to make them more manageable. Similarly, in another study Nejmeh [1988], based on the experience, the author declared the threshold value of NPATH as 200.
The idea of the application of fixed threshold does not seem reliable in all settings. There exists a variety of software systems each of which is of different size and possesses other distinct features. The application of fixed thresholds in the software systems holding distinct characteristics does not seem to yield efficient results. In this case, some statistical methods are required, which can calculate the final thresholds after analyzing the characteristics of the specific software. Many researchers followed this approach and tested different statistics in the realm of fault prediction to deliver the appropriate thresholds. In Erni and Lewerentz [1996] and French [1999], the authors adopted most commonly utilized quantitative method, i.e. mean and standard deviation, to produce the threshold values. These methods were not formulated with aim to group the software classes based on the fault status. But, later on, the method suggested by Erni and Lewerentz [1996] was utilized by another researcher to distinguish the classes on the ground of faults, with the objective to aid the decision making process of distributing the testing efforts systematically Shatnawi [2015].
The Value of an acceptable risk level (VARL) is the most assessed technique for its classification ability in the realm of fault prediction Shatnawi [2010], Singh and Kahlon [2014], Malhotra and Bansal [2015]. Originally, this method was discovered in an epidemiology field, along with another method- value of an acceptable risk gradient, to identify the relationship between glycosylated hemoglobin and diabetic nephropathy development Bender [1999]. Shatnawi [2010] introduced the VARL statistic in the field of fault prediction and evaluated its discrimination ability by computing the metric thresholds of eclipse 2.0 and tested the resultant values by grouping the software classes of the subsequent release. The outcome showed the effectiveness of the statistic in grouping the classes into low-risk and high-risk categories. But the author found different

results in the case of other software systems, Mozilla and Rhino, where the technique failed to give appropriate thresholds for CK metrics. Afterwards, Singh and Kahlon tested the proficiency of VARL technique in calculating the thresholds of OO metrics- DIT, NOC, RFC, WMC, LCOM, Co, CBO, NOA, NOOM, NOAM, PuF, and EncF Singh and Kahlon [2014]. The authors executed the entire experiment on three versions of Mozilla Firefox - 1.5, 2.0, and 3.0. Among twelve metrics, the thresholds of only four metrics (CBO, RFC, WMC, EncF) were found proficient in identifying the faults. Malhotra and Bansal [2015] also examined the effectiveness of VARL technique to find the thresholds of CK metrics. The authors also performed inter-project fault prediction, where they retrieved the metric thresholds of Ivy and applied the computed thresholds on Tomcat and Ant. In another study Arar and Ayan [2016], the VARL method was applied to determine the metric thresholds that can discriminate the classes into three categories- non-faulty classes, faulty classes with one fault and faulty classes containing more than two faults.

Shatnawi et al. [2010] employed Receiver Operating Characteristic (ROC) curve to find the threshold values of 12 OO metrics. The authors carried out the experiment on three releases of eclipse software- 2.0, 2.1 and 3.0. They categorized the modules on binary and multinomial levels. In case of binary categorization, the identified thresholds failed to categorize the eclipse modules into faulty and not-faulty categories. However, Boucher and Badri [2017] tested the same technique on the distinct datasets, and proved the potency of ROC curve in carrying out the binary classification. In the case of multinomial categorization, Shatnawi et al. [2010] tried to classify the modules into three levels (low, medium and high), but better results were obtained after combining the medium and high categories.

A.M.Ferreira et al. [2012] put forward another threshold technique that works by fitting the data to different probability distribution, where, Poisson and Weibull distributions were identified to be the best fit. The recommended method was used to collect the thresholds of Coupling Factor, Number of Public Fields, Number of public methods, Lack of Cohesion methods, Depth of inheritance tree, and Afferent coupling. Based on the retrieved thresholds, the classes were categorized into three levels, i.e., good, regular and bad, which in turn helped to decide how well the design principles were followed by a particular class.

Shatnawi [2015] proposed the idea of transforming the dataset into normal distribution form before the application of standard deviation plus mean method for threshold calculation. On the grounds of the information presented in the study Basili et al. [1996], Shatnawi claimed that the presence of skewed data can adversely affect the threshold outcome. In order to obtain the reliable results, there is the need to make the data to follow the normal distribution pattern. Consequently, Shatnawi applied log transformation on the selected metrics and then calculated the threshold values. Subsequently, the calculated threshold values were transformed back to the original form by using an exponential formula. And, at last the computed thresholds were applied to the classes following skewed distribution. The author also performed the binary classification by using the thresholds retrieved from the application of standard deviation plus mean method on the skewed dataset. After this, the predictive results obtained in case of skewed dataset were compared with the results of transformed dataset, where the latter approach gave significantly better results.

Alves et al. [2010] proposed another threshold technique, capable of producing appropriate thresholds even in case of skewed dataset. This technique involves the execution of six steps to compute the final threshold. This technique was originally tested on the method-level metrics, later on, it was also examined in Boucher and Badri [2017] on CK metrics and was named as Alves Rankings. The authors compared the prediction accuracy of three threshold techniques, i.e. ROC curve, VARL and Alves Rankings Boucher and Badri [2017]. The results showed the poor predictive capability of VARL technique in identifying the true thresholds. ROC curve gave the best results followed by Alves Rankings technique. In addition, the models constructed using threshold techniques were also compared with machine learning approaches such as Bayes Network, Artificial Neural Network, C4.5, Support Vector Machine, K-means, and Self Organizing Map. The find-

ings showed that machine learning techniques when integrated with threshold techniques provides better results as compared to outcome solely based on the machine learning techniques.
This study is motivated by the work of Boucher and Badri [2017] and performs the comparison of the predictive capability of ROC curve and Alves Rankings. This study is different from the previous one in terms of number of metrics considered in the evaluation process. It is necessary to mention here that Boucher and Badri also considered the VARL statistic in their study, but VARL technique was unable to provide valid values for most of the software systems, on the other hand the systems which managed to give the valid threshold values were showing very poor predictive results. Therefore, we decided to omit the inclusion of VARL in this study.

3. RESEARCH METHODOLOGY

The main motive of this study is to investigate the concept of threshold that can discriminate the software classes based on the threshold values, where the classes with metric values greater than the threshold are conceived to be faulty and classes with values lower than the threshold are conceived to be non-faulty. This section contains the methodology adopted to probe the objective of this study.

3.1 Data Collection

In the field of fault prediction models, metric data and fault information are two mandatory elements which make the execution of research possible. Based on the availability of these elements, the researchers classified the dataset into three categories- public, private and partial Radjenovi et al. [2013]. In public dataset, both of the metric data and fault information are available. In partially public datasets, the source code of software system is available and metric data can be computed from the code. Many tools are available to purport the metric value extraction process. In private dataset, neither metric data nor fault information is available. The present study used the public dataset to assess the capability of selected threshold methods to distinguish the faulty and non-faulty classes. Basically, the dataset utilized in the study [29] are adopted to execute the experiment. The authors of the mentioned study uploaded the dataset on the GitHub[1] . The current study exercises the concept of 10 fold cross-validation to train and test the threshold techniques. The size of the dataset on which the cross-validation is to be applied should be sufficiently large. Among the 30 available software systems, only 14 of them were meeting the size standards, therefore, the rest was omitted. For the cross-version fault prediction experimentation, we utilized the dataset available at tunedit [2] repository . The software systems selected from the aforementioned public domains are listed in Table I, where the systems with Sr. No. 1-14, were employed in experiment of cross-validation, whereas the remaining two software systems were used for cross-version fault prediction experimentation. Table-1 Details of dataset considered for the experiment.

As already mentioned, this study is motivated by the work of Boucher and Badri [2017], and some of the software systems investigated in the current study (Ant-1.7, IVY, Lucene, POI, tomcat, and jEdit) have already been examined by the study under reference. But, the authors of the current study have also derived the thresholds of other OO measures, in addition to CK metrics. The inclusion of additional measures makes this study different from the previous one. Furthermore, Boucher and Badri executed the idea of cross-version fault prediction on the software system Ant, whereas, the authors of the current study performed the same experiment on distinct software system, where the threshold values extracted from jEdit 4.2 was tested on its subsequent version (jEdit-4.3).

[1]https://github.com/lov505/JSS-Dataset/Processing.rar
[2]http://tuneDIT.org/

S_ID	Dataset	No. of classes	No. of faulty Classes	Percentage of faulty classes
S1	Ant-1.7	745	166	22%
S2	Arc	234	27	11%
S3	Camel-1.6	965	188	19%
S4	Ivy-2.0	352	40	11%
S5	Log4j-1.2	205	189	92%
S6	Lucene-2.4	340	203	59%
S7	Poi-3.0	442	281	63%
S8	Prop-6	660	66	10%
S9	Redactor	176	27	15%
S10	Synapse-1.2	256	86	33%
S11	Tomcat	858	77	9%
S12	Velocity-1.6	229	78	34%
S13	Xalan-2.7	909	898	98%
S14	Xerces-1.4	588	437	74%
S15	jedit-4.2	275	134	49 %
S16	jedit-4.3	275	204	75%

Table I: Details of dataset considered for the experiment

3.2 Dependent and Independent variables

In this study, fault-proneness of a class is considered as a dependent variable and software metrics are considered as independent variables. Basically, the threshold technique can categorize a set of software classes into predefined groups based on the threshold values derived from the independent variable. In the prior fault prediction studies, three types of classifications have been performed- binary, multinomial and fault density. In binary classification, the modules are classified into either faulty or non-faulty groups. In multinomial classification, the modules are categorized based on the severity of the error, i.e. low, medium, and high-severe faults. And in the fault density, classifier predicts the number of faults that are likely to occur in each module during the maintenance phase. In this study, the dependent variable is of binary nature, i.e. the output of the metric threshold categorizes a class either into faulty or non-faulty group. The class with no fault is considered as non-faulty, whereas, the class containing one or more number of faults is considered as a faulty class.

Both industrial and academic experts carried out an extensive research to identify different software measures. The usefulness of the proposed metrics was further proved by the researchers by utilizing them for evaluating different quality attributes of the software system. The aim of this study is to identify the threshold values of OO metrics that can be helpful in detecting the faulty classes. The success of the final model heavily depends on the software measures selected for predicting the fault-proneness. Thus, careful selection of the software metrics is highly required, which are to be used for the threshold calculation. Therefore, only the selective metrics which were claimed to have prognostic capability of identifying faulty entities are chosen in the current study. But, before calculating the threshold value of the considered metric, it is mandatory to confirm the presence of a significant relationship between fault information and selected measures.

The software measures suggested by Martin et al., Bansiya and Davis, Halstead, and Hendersonseller are selected for the current work. Table II contains the list of measures considered to execute the current study. All of these metrics fall into one of the five categories, i.e. size, coupling, complexity, cohesion, and inheritance. The metrics NPM, LOC, DAM, MOA, and MFA come under the size category. Furthermore, metrics CBO, RFC, Ca, Ce, and CBM represent the measures capturing coupling information. Similarly, metrics LCOM, LCOM3, and CAM captures cohesion information. And last, metrics WMC, AMC, MAX_CC, and AVG_CC retrieve complexity related information from the source code. The researchers recommend the low values

of all metrics, except the cohesion, in order to make the software more manageable and error free.

Authors	Metrics
Chidamber and Kemerer [1994]	Weighted method per class (WMC), Depth of Inheritance Tree (DIT), Number of children (NOC), Coupling Between Object classes (CBO), Lack of Cohesion in Methods (LCOM), Response For a class (RFC)
Martin. [1994]	Afferent Coupling (Ca), Efferent Coupling (Ce)
Bansiya and Davis [2002]	Number of Public Methods (NPM), Measure of Aggregation (MOA), Measure of Functional Abstraction (MFA), Cohesion among Methods of class (CAM), Data Access Metric (DAM)
Tang et al. [1999]	Coupling between Methods (CBM), Average Method Complexity (AMC), Inheritance coupling (IC)
Henderson-Sellers [1996]	LCOM3
McCabe [1976]	Maximum Cyclomatic Complexity(Max_CC) , Average Cyclomatic Complexity(Avg_CC)
Halstead [1977]	Lines of Code(LOC)

Table II: The OO metrics considered for threshold calculation

3.3 Univariate analysis

The metrics selected to train the classifiers have an enormous impact on their predictive accuracy. Before the inclusion of the selected measures in the prediction models, their prognostic capability of detecting faulty classes needs to be confirmed. Irrelevant measures need to be excluded from the model to produce the quality results. In the current study, the metric selection step is also mandatory to discard the measures that paid scant contribution in anticipating the fault-proneness of the class. Different techniques have been utilized in the literature to identify the appropriate set of software measures for the selected software systems. Basili et al. [1996] conducted empirical investigation of OO measures toward fault prediction. Before the formulation of the multivariate logistic model, the authors utilized Univariate Logistic Regression (ULR) to identify the predictive capability of the selected metrics. In another study lLov Kumar et al. [2018], ten distinct techniques were employed to find the OO measures efficient for inclusion in the Least Square Support vector Machine (LSSVM) model. The selected techniques were- Chi Squared test, gain ratio feature evaluation, information gain feature evaluation, oneR feature evaluation, logistic regression analysis, principal component analysis, correlation based feature selection, rough set analysis, consistency subset evaluation technique, and filtered subset evaluation.

The authors of the current study have employed ULR to select the proficient metrics having significant level of association with the fault-proneness. In this technique, each of the metrics is evaluated in the context of dependent variable to check their predictive efficiency.

3.4 Identification of threshold values

The selection of threshold value needs to be done very precisely. When the selected threshold value is lower than the exact cut-off point, then there will be a high number of false positive rate, similarly when this value is greater than the appropriate threshold, then there will be a high number of false negative rate. Therefore, the selection of exact threshold is essential to produce the accurate final results. The subsequent subsections explain the detailed methodology of the selected threshold techniques.

3.4.1 *ROC curve.* Analysis using ROC curve initially took place in radar technology Erdreich and Lee [1981], which later on also gained popularity in other fields, such as, laboratory medicine and social sciences. In the area of Software Engineering, the most popular use of ROC curve is to evaluate the prediction accuracy of the models developed using statistical and machine learning

algorithms. The ROC measure is present as an inbuilt function in many of the well-known software packages and programming languages, such as, Weka, SPSS, Python, Matlab, and R. In the domain of fault prediction using threshold construct, ROC was first introduced by Shatnawi et al., to discriminate the classes into high-risk and low-risk categories. In ROC plot, the x-axis represents the 1-specificity and the y-axis represents the sensitivity. An optimal cut-off on the ROC curve can be identified by maximizing the selected function of sensitivity and specificity. The entire process of threshold calculation is divided into two steps:

a) The first step involves the construction of the ROC curve. For this, each value in the metric range is taken as the threshold and then the prediction is conducted based on the selected threshold, where, the classes with metric value higher than the selected point is considered as faulty, otherwise, non-faulty (except the cohesion measure). The confusion matrix indices (as shown in Table III) are calculated using the prediction results based on the selected threshold. Further, the computed indices help to find the sensitivity (also known as True Positive Rate (TPR)) and 1-specificity (also known as False Positive Rate (FPR)). Sensitivity is the proportion of faulty classes that are correctly measured as faulty (formula given in equation (1)) and specificity is the proportion of non-faulty classes that are correctly measured as non-faulty (formula given in equation (2)). Here, the ROC curve uses 1-specificity, which basically tells the proportion of fault-free classes that are incorrectly classified as faulty classes. The deduced values of sensitivity and 1-specificity help to plot a point on the XY plane. This procedure needs to be repeated for every value in the metric range and, at last, a curve will be formed from all of the deduced points.

Predicted	**Actual faulty**	**Actual non-faulty**
Value>=Threshold	True Positive	False Positive
Value<=Threshold	False Negative	True Negative

Table III: Confusion matrix

$$Sensitivity = TruePositive/(TruePositive + FalseNegative) \tag{1}$$

$$Specificity = 1 - FalsePositive/(FalsePositive + TrueNegative) \tag{2}$$

b) Second step incorporates the identification of the optimal cut-off point on the ROC curve, which can unerringly discriminate the positive and negative classes. The selection of an optimal point requires the trade off between TPR and FPR. There are a number of methods available to find the optimal point on the ROC, such as, selection of point with maximized Youdens index value, the point closest to upper left corner of the ROC plot, the point with maximized area under curve, and many more. In this study, we opted Youdens index (YI) method to identify the optimal cut-off point Youden [1950]. The aforementioned method was initially suggested in the medical field by Youden. Basically, YI is a function of sensitivity and specificity, such that:

$$YoudenIndex = Sensitivity(p) - (1 - Specificity(p)) \tag{3}$$

The value of YI is computed for each point p on the curve and the point having a maximum value of YI is considered as the optimal point on the ROC curve.

3.4.2 *Alves Rankings.* Alves et al. [2010] suggested a threshold derivation technique capable of identifying an optimal cut-off point which can efficiently group the classes into low-risk and high-risk categories. Originally, the authors tested this technique on method-level metrics, but later on, also got experimented on OO metrics by Boucher and Badri [2017]. The threshold model trained through this technique does not require the availability of the fault information, thus, prove their superiority over ROC curve. The identification of metric thresholds through the selected technique requires the execution of following six steps.

a) The first step includes the extraction of the metric value for each entity (in our case, the entity is a class) present in the system. Here, we need to choose a metric (for example WMC) of which the threshold value is to be calculated along with weight metric, i.e. LOC.

b) The next step includes the calculation of the weight ratio. This step requires the division of weight, corresponding to each class, with the total weight of all classes in the system. For example, if the weight of class A in Software S is 1000 LOC and total size of S is 10K, then the weight ratio for the class A will be 10%.

c) The third step is the entity aggregation. It requires the summation of weight ratios of the classes having similar value of the selected software measure. Suppose the values of WMC metric of six classes A, B, C, D, E and F in software system S are 2, 3, 1, 3, 5, and 2. And, the weight ratios of these classes are 20%, 10%, 15%, 30%, 10%, and 15%, respectively, then the result after entity aggregation will be 35% in case of 2 metric value, 40% in case of 3, 15% in case of 1, and 10% in case of 5 metric value.

d) The next two steps involve the system aggregation and weight ratio aggregation. These steps are omitted from the current paper as here each system is considered independent of other systems, therefore, the technique will be applied to each software system separately, as opposed to Alves et al. [2010] where the authors produced collective results of hundred software systems by aggregating the metric values as defined in the previous step.

e) The last step involves the threshold derivation. It involves the construction of a cumulative line chart and selection of the metric value as a threshold corresponding to the specific weight ratio which give efficient results, Alves et al. considered the metric values, at weight ratios 70%, 80%, and 90%, as the optimal threshold values.

3.5 Cross-version and cross-project fault prediction

As already mentioned, we picked 14 software systems to validate the threshold techniques. In order to accomplish the objective, relevant data is required to compute the threshold values and different data is required to validate the computed values. This process of training and testing needs to be conducted on the data of the same software system. Many machine learning classifiers have served this purpose through the concept of cross validation. Also, in the area of fault prediction, cross validation has been employed to get more realistic results by splitting the dataset into training and testing parts Malhotra et al. [2011], Malhotra and Singh [2011], Malhotra and Jain [2012]. During the experiment, we adopted 10-fold cross validation to achieve the required objectives of determining and testing the threshold values. Besides this, we also tested the validity of threshold construct on the cross-version and cross-project fault prediction.
Cross-version fault prediction: In this case, the threshold values retrieved from a software release are tested on its subsequent release. The selected source, i.e., tunedit repository contains the metric information of the different versions of the software system. Among the available software systems, we selected the JEdit to answer the question, *Whether the metric thresholds calculated from one version of the software system are helpful in discovering the faults in its subsequent version?*
Cross-project fault prediction: The selection of different software systems for sharing the threshold values is itself a major issue. The thresholds extracted from one environment might not produce the efficient results in every environment. In this case, the demand for the identification of software systems having some similar feature arises, so that the threshold values can be reused across these identified software systems. To achieve this motive, we delimited two conditions to select the software systems based on their internal characteristics (discussed in the next section). The systems satisfying these conditions were put into one category and threshold values were shared among these systems to validate the idea of cross-project fault prediction. Basically, this experiment was conducted to answer the question, *Whether the metric thresholds calculated from one project are helpful in predicting the faults of other projects?*

3.6 Performance evaluation measures

A range of performance evaluation measures has been utilized by researchers to verify the predictive efficiency of the developed prediction models Jiang et al. [2008]. In our case, the proportion of faulty and non-faulty classes in most of the selected software systems is highly imbalanced, where the percentage of faulty classes clearly outnumbered the non-faulty classes (except the Xalan-2.7). Therefore, the application of evaluation measures, such as accuracy and log-loss, is not an appropriate choice here, as the aforementioned measures are suitable only in case of balanced dataset. By taking into account the above aforementioned fact, the measures considered for assessing the predictive capability of the investigated threshold techniques are False Negative rate (FNR), False Positive rate, sensitivity, specificity and G-mean. Boucher and Badri [2017], also employed the same measures for evaluating the outcome of threshold and machine learning based prediction models .

The FNR is the proportion of faulty classes that are incorrectly predicted as fault-free, whereas, FPR is the proportion of non-faulty-classes that are incorrectly predicted as faulty. The formula of FPR and FNR is given in equation (4) and (5). Both FNR and FPR indices of the constructed prediction model should be low. In some domains, the costs associated with the harms of false positives and false negatives can be very hazardous, for e.g. in the medical field, if the model is constructed for predicting some deadly disease then their high false negative results can be proved as a menace to human life. Also, in the era of fault prediction, the value of FNR should be low, as the classifier producing high FNR will predict the faulty classes as non-faulty, as a result, limited or no testing efforts will be devoted to faulty classes. The consequences of which most of the errors will remain undetectable before the delivery of the final product, which will directly impact the future maintenance cost. On the contrary, in case of high FPR, the limited testing efforts will be expended upon the bug free classes, which may lower the overall efficiency.

$$FPR = FalsePositive/(FalsePositive + TrueNegative) \tag{4}$$

$$FNR = FalseNegative/(FalseNegative + TruePositive) \tag{5}$$

Other measures used in the study are sensitivity, specificity, and G-mean (formulae given in the equations (1) and (2) and (6)). The computation of G-mean requires both sensitivity and specificity indices. Kubat and Matwin suggested G-mean method, capable of measuring the balance between classification performances on both the negative and positive cases Kubat and Matwin [1997]. The higher value of the G-mean method signalizes the expertise of computed threshold in delivering the outstanding results, on contrary their low value indicates the poor predictive performance. The classifier will give low G-mean value even if it successfully identified all negative cases, but failed to identify the positive ones. As G-mean gives a final result by considering both the sensitivity and specificity indices, thus, in the result section we will not separately show the outcome of these two indices.

$$G - mean = \sqrt{sensitivity * specificity} \tag{6}$$

3.7 Method for comparing threshold techniques

To identify the best threshold technique between ROC curve and Alves ranking, some comparison technique is required that can efficiently act in the scenario where the techniques are to be compared over multiple datasets. One solution is the calculation of the average performance of each technique on all available systems and then selection of the technique with better performance results. But, Demsar [2006] proved their associated drawbacks and suggested the application of Wilcoxon signed rank test when two techniques need to be compared over multiple datasets. Demsar also mentioned the use of Friedman test when the number of techniques to be compared is more than two. Boucher and Badri [2017], in their study, used the Friedman test along with post-hoc Nemenyi test .

The authors of the current study selected the Wilcoxon signed rank test to compare the predic-

tion efficiency of ROC and Alves Rankings. The application of this test is also observed in other related fault prediction studies where researchers employed this test to compare different machine learning classifiers Elish [2012], Kumar et al. [2017]. The Wilcoxon test is non-parametric test, which does not assume the normality in data distribution, as opposed to paired sample t-test which inversely assumes the normal distribution of the variables. The null hypothesis of Wilcoxon method states the equality between the performances of the compared methods, whereas, the alternate hypothesis states their inequality. For each dataset, the difference between the performances of the examined techniques is computed. The ranks are assigned to the computed differences based on their merit by ignoring the signs and consequently these ranks are compared for the positive and negative differences. Assume the size of dataset is N_{ds}, the Wilcoxon test can be computed by using the formula (7) and (8), where, d_i denotes the difference between the performance measures of two prediction techniques on i^{th} dataset, and R^+ denotes the summation of ranks of the datasets where first technique shows better results than other technique, and R^- denotes the summation of the remaining datasets. According to this method, the null hypothesis will be rejected, if the difference between the sum of the positive and negative ranks is less than or equal to the value of the distribution of the Wilcoxon for N_{ds} degrees of freedom, otherwise it will be accepted Zar [1999].

$$(R)^+ = \sum_{d_{i>0}} rank(d_{i>0}) + \frac{1}{2}\sum_{d_{i=0}} rank(d_i) \tag{7}$$

$$(R)^- = \sum_{d_{i>0}} rank(d_{i<0}) + \frac{1}{2}\sum_{d_{i=0}} rank(d_i) \tag{8}$$

The comparison of the outcomes of methods A, B, and C is illustrated in the Table IV. The comparison procedure starts with the calculation of positive and negative ranks for each of the dataset by using the equations (7) and (8) respectively. Positive ranks contain the summation of differences between the performance outcomes of methods A and B, of those datasets where method A produced better results. On the contrary, negative ranks contain the summation of sum of differences of the datasets where method B performed better than method A. In the table, the yielded value of negative ranks evident the superiority of method B over the method A. However, before declaring A as winner method, it is essential to verify whether the observed difference between the calculated ranks is actually significant. The significance of difference can be assured from the p-value (which should be below than 0.5) and the resultant p-value indicates the supremacy of method B over method A. In Table 4, the boldface methods are winner.

Methods	**Positive ranks**	**Negative ranks**	***P*-value**
A-**B**	0.6	101.2	.001
A-C	.85	.15	.021

Table IV: Wilcoxon signed rank test to identify the best method

4. RESULTS

The following subsections contain the detailed results obtained during the experiment. Here, the experimental results about the level of association existing between the dependent and independent variables are reported first. After which the results about the threshold values and their predictive efficiency is discussed in the very next subsections. At last, the prognostic capability of the considered techniques is compared.

4.1 Univariate Logistic Regression

One of the most commonly used methods to analyze the effectiveness of the OO measure toward anticipating fault-proneness of class is ULR. Tang et al. [1999] employed URL method and calculated the values of the coefficients, p-value, R^2, and odds ratio, in order to select the metrics showing a significant relationship with fault-proneness. The authors considered the association between the variables as statistically significant if its p-value is below 0.1. Malhotra and Bansal [2015] also used ULR and calculated the parameters, such as, coefficient, constant, and statistical significance, for each of the metrics. The metrics with p-value less than .05 was selected for the further analysis. Boucher and Badri also used ULR but considered different parameters, such as p-value, Wald chi-square, and R^2, to identify the effectiveness of individual metrics toward fault-proneness. The relationship with p-value below .05 was viewed as significant and selected the metrics having required p-value. The authors omitted LCOM metric from further experiment due to their low values of R^2 and Wald chi-square. The ULR was executed on 12 datasets, and the metrics found significant in at least 9 of the total datasets were chosen for the further investigation. The selected metrics were SLOC, CBO, RFC, and WMC. The current study selected URL to verify the existence of the required relationship between the software measure and binary outcome. The main objective of employing this method is their ability to handle the situation where the dependent variable is of dichotomous nature, i.e. output is either 0 or 1. Here, the relationship between the independent and dependent variable is considered as significant if its p-value is below .05. As the metric and fault information of 14 software systems have been utilized in the experiment, here only those metrics that were discovered statistically significant in more than ten software systems have been chosen for further analysis. As shown in the Table V, out of 14 the WMC was found significant in 13 systems, DIT in 1, NOC in 3, CBO in 13, RFC in 14, LCOM in 8, Ca in 5, Ce in 12, NPM in 12, LCOM3 in 9, LOC in 13, DAM in12, MOA in 12, MFA in 2, CAM in 12, IC in 9, CBM in 9, AMC in 11, Max_CC in 10, and Avg_CC in 9 software systems. The observations revealed that three complexity metrics (WMC, AMC, and MAX_CC), three coupling measures (CBO, RFC, and Ce), one cohesion measure (CAM) and all of the size metrics (NPM, LOC, DAM, and MOA) showed the significant relationship with the binary outcome. The findings disclosed the disassociation of inheritance metrics with fault-proneness. In summary the important findings of this section are:

—The metrics DIT, NOC, LCOM, Ca, LCOM3 MFA, IC, CBM, and Avg_CC showed insignificant ability in discriminating the faulty and non-faulty classes.

—Whereas, the metrics that showed significant association with fault-proneness and got selected for further investigation are WMC, CBO, RFC, Ce, NPM, LOC, DAM, MOA, CAM, AMC, and Max_CC.

4.2 Calculation of threshold values

This subsection includes the application of the threshold techniques on the metrics selected in the previous subsection in order to retrieve their threshold values. The entire process of threshold extraction and validation, using ROC curve and Alves Rankings, was implemented in the Python programming language. Based on the threshold values acquired during the training phase, the classification process was executed on the testing dataset, where for all metrics, except CAM, the classes having metric value more than the threshold point were put under the faulty category. The past studies emphasized on the high cohesiveness in the class as its low value directly impacts the quality of the software systems. In case of CAM, if the class value is more than threshold point, then it will be considered as non-faulty, otherwise faulty.

4.2.1 *Threshold extraction using ROC curve.* The ROC curve is constructed by plotting the sensitivity against 1 minus specificity for each of the metric values. Any point on the curve can be considered as a threshold, but determination of exact point is essential to produce satisfactory results. The identification of exact threshold value needs the trade-off between true positives and

Metrics	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14
WMC	.000	.001	.000	.000	.045	.000	.000	.000	.150	.000	.000	.000	.042	.017
DIT	.332	.230	.304	.792	.974	.118	.195	.097	.067	.608	.452	.124	.072	.000
NOC	.529	.571	.000	.450	.997	.027	.811	.448	.365	883	.079	.085	.997	.043
CBO	.001	.001	.000	.000	.018	.000	.006	.000	.370	.000	.000	.000	.029	.000
RFC	.000	.000	.000	.000	.138	.000	.000	.000	.015	.000	.000	.000	.010	.000
LCOM	.000	.130	.000	.000	.226	.001	.000	.000	.401	.205	.000	.021	.186	.172
Ca	.186	.263	.000	.021	.068	.000	.347	.150	.336	.266	.000	.326	.338	.000
Ce	.000	.000	.003	.000	.316	.000	.000	.000	.002	.000	.000	.000	.031	.000
NPM	.000	.007	.000	.000	.027	.000	.000	.000	.687	.002	.000	.000	.000	.895
Lcom3	.001	.935	.000	.076	.168	.000	.000	.082	.632	.000	.002	.003	.016	.000
LOC	.000	.000	.000	.000	.395	.000	.000	.000	.000	.000	.000	.010	.000	.000
DAM	.000	.613	.000	.001	.009	.000	.000	.002	.302	.000	.000	.025	.034	.000
MOA	.000	.003	.001	.000	.677	.011	.000	.000	.007	.000	.000	.000	.406	.000
MFA	.587	.082	.574	.247	.620	.948	.814	.133	.001	.788	.443	.087	.084	.000
CAM	.000	.004	.000	.000	.092	.000	.000	.000	.199	.000	.000	.000	.037	.000
IC	.000	.182	.166	.565	.343	.000	.000	.036	.006	.028	.001	.031	.121	.000
CBM	.001	.169	.022	.105	.197	.003	.000	.124	.000	.005	.000	.030	.180	.000
AMC	.201	.003	.010	.001	.060	.013	.030	.047	.000	.000	.000	.000	.182	.000
Max_CC	.000	.031	.000	.000	.302	.118	.001	.002	.039	.000	.000	.156	.325	.000
Avg_CC	.000	.048	.000	.000	.227	.402	.000	.037	.487	.043	.000	.449	.682	.000

Table V: Threshold values calculated using ROC curve

false positives. In certain instances, true positive rate is of higher importance than false positive rate, for example, when a fault has catastrophic impact on the performance of the system. On the contrary, in some situations, the false positive rate is preferred to true positive rate, say when the cost of testing efforts is very high, in this situation the number of false positives should be very low. But, if no such choice exists between the true positive and false positive rate, then an acceptable approach is the selection of optimal point on ROC curve by using any of the methods (as discussed in section 3.4.1). The threshold values obtained through the application of ROC curve are shown in Table VI. The value of Ce metric is 0 for all of the classes of the Tomcat software system, so we marked this cell with an asterisk sign (*) in Table VI and Table VII.

S_ID	WMC	CBO	RFC	Ce	NPM	LOC	DAM	MOA	CAM	AMC	Max_CC
S1	10	7	32	5	7	178	0.058824	0	0.360000	15.750000	4
S2	6	12	35	2	6	132	0.933333	0	0.406250	12.333333	2
S3	9	9	9	3	5	148	0.250000	0	0.329670	7.642857	1
S4	7	7	38	6	13	277	0.0	1	0.400000	18.000000	3
S5	2	5	37	4	2	15	0.0	1	0.666667	9.300000	1
S6	5	5	15	3	5	106	0.142857	0	0.340000	5.666667	2
S7	9	5	17	3	7	95	0.052632	0	0.333333	7.625000	1
S8	10	8	38	9	9	229	0.727273	0	0.346667	5.142857	4
S9	15	16	35	6	13	314	0.0	0	0.714286	28.600000	1
S10	5	13	31	11	5	200	0.0	0	0.458333	14.750000	4
S11	13	6	38		16	375	0.166667	1	0.431818	16.344828	7
S12	4	8	25	3	4	94	0.250000	0	0.457143	17.500000	3
S13	3	3	5	2	1	67	0.500000	2	0.500000	7.000000	3
S14	3	1	7	0	3	26	0.0	0	0.638889	4.250000	1

Table VI: Threshold values calculated using ROC curve

4.2.2 *Threshold extraction using Alves Rankings.* As discussed earlier, Alves et al. [2010] suggested a series of steps to calculate the threshold values. The retrieval of threshold through this technique requires the selection of appropriate weight ratio, as the metric value corresponding to this weight ratio will be selected as an appropriate threshold value. Alves et al. chose the

metric values at weight ratio 70%, 80%, and 90% as the appropriate threshold values. In the current study, the predictive results obtained at weight ratios 10%, 20%, 30% and up to 100% were analyzed and findings indicated that better predictive results were appearing at the weight ratios 20% and 30%. In some cases, metric values at 20% were showing good results, whereas in other cases, the values at 30% were revealing the better results. Table VII shows the threshold values obtained with the application of Alves Rankings at weight ratio 20% and 30%. From the threshold results it can be observed that in case of LOC measure, the great variation can be observed between the thresholds of ROC and Alves techniques in Log4j-1.2, Poi, Velocity, Xalan, and Xerces. Similarly, in case of MOA measure, the threshold values derived for all of the software systems are 0, which are different from the thresholds of ROC curve. And, the threshold values of CAM measure identified through Alves Rankings are smaller than the ROC curve in all of the software systems.

S_ID	WMC	CBO	RFC	Ce	NPM	LOC	DAM	MOA	CAM	AMC	Max_CC
S1(20%)	8	5	30	3	5	221	0.583333	0	0.155080	17.50000	2
S1(30%)	12	7	40	5	7	322	0.800000	0	0.178947	20.65000	4
S2(20%)	9	5	22	2	8	122	0.933333	0	0.138315	10.33330	1
S2(30%)	13	7	29	5	11	172	0.933333	0	0.148607	13.77778	2
S3(20%)	6	7	20	4	3	104	0.833333	0	0.160000	8.750000	1
S3(30%)	8	8	27	6	4	145	0.954545	0	0.109557	10.25000	1
S4(20%)	8	9	38	0	4	242	0.666667	0	0.163194	15.857143	2
S4(30%)	12	12	59	2	6	321	0.894737	0	0.163194	17.666667	3
S5(20%)	6	3	20	1	2	157	0.0	0	0.163265	15.800000	2
S5(30%)	7	5	28	3	3	202	0.187500	0	0.203947	18.882353	3
S6(20%)	8	4	20	2	3	278	0.210526	0	0.139194	20.745098	2
S6(30%)	12	6	28	3	5	389	0.400000	0	0.178571	26.166667	3
S7(20%)	9	5	26	2	7	240	0.142857	0	0.180000	13.270270	2
S7(30%)	11	6	31	3	9	323	0.444444	0	0.190476	15.222222	2
S8(20%)	5	6	19	1	2	134	0.954545	0	0.183908	11.500000	2
S8(30%)	6	8	29	4	3	177	0.954545	0	0.225000	14.000000	3
S9(20%)	8	8	26	4	5	284	0.857143	0	0.246377	25.636364	2
S9(30%)	10	10	30	6	6	341	0.857143	0	0.277778	28.700000	3
S10(20%)	4	9	27	9	2	174	0.714286	0	0.196429	21.000000	4
S10(30%)	5	11	31	10	3	220	0.967742	0	0.255102	25.312500	7
S11(20%)	9	3	27	*	4	248	0.666667	0	0.137006	20.333333	2
S11(30%)	13	6	44	*	6	543	0.909091	0	0.167155	25.333333	4
S12(20%)	6	4	28	3	5	239	0.333333	0	0.145559	22.600000	2
S12(30%)	9	5	38	4	6	342	0.382353	0	0.236111	28.200000	43
S13(20%)	3	1	14	0	2	425	0.0	0	0.180871	25.268116	1
S13(30%)	4	2	14	1	2	666	0.0	0	0.232143	34.986110	2
S14(20%)	8	4	22	1	3	304	0.0	0	0.177778	22.100000	2
S14(30%)	12	6	32	2	5	558	0.142857	0	0.187500	28.840909	3

Table VII: Threshold values calculated using Alves Rankings

4.3 Performance results

Choosing an appropriate threshold technique is of paramount importance in order to acquire the inerrant prediction results. Several performance evaluation measures have so far been employed in choosing the most suitable prediction model. In the study Shatnawi [2010], the authors used G-mean method for evaluating the performance of VARL technique, whereas, in another study Singh and Kahlon [2014], the ROC curve was used to test the effectiveness of thresholds computed through the VARL technique. In the study Malhotra and Bansal [2015], the authors used sensitivity, specificity, G-mean, and AUC measures to ensure the predictive potentiality of thresholds calculated using VARL technique. Arar and Ayan [2016] used true positive rate, true negative rate, and G-mean for appraising the anticipating capability of the selected threshold

technique.
In the current study, we computed the values of FPR, FNR, and G-mean measures to assess the prognostic capability of the cut-off values acquired using ROC curve and Alves techniques. The authors of the study Boucher and Badri [2017] classified the predictive capability of the threshold model into one of the five categories on the basis of their retrieved G-mean value. The current study also followed the similar classification approach in which the threshold techniques were classified into one of the five categories as given in the Table VIII. The prerequisite of computing

G-mean	**Classification**
Below 0.5	No good classification
Between 0.5 and 0.6	Poor classification
Between 0.6 and 0.7	Average classification
Between 0.7 and 0.8	Acceptable classification
Between 0.8 and 0.9	Excellent classification
Greater than 0.9	Outstanding classification

Table VIII: Classification of techniques based on G-mean value

the threshold value through Alves Rankings is the selection of the appropriate weight ratio Alves et al. [2010]. Therefore, this weight ratio should be chosen very wisely. Alves et al., in their study, selected the metric values corresponding to weight ratios 70%, 80%, and 90%. The classes containing the metric values less than the threshold at weight ratio 70% were put into the low-risk group. Similarly, the classes with metric values range between the thresholds at 70% and 80% were put into the moderate-risk group, the metric values in the range between 80% and 90% were grouped into high-risk classes and the values above 90% were put into very high-risk classes. On the other hand, Boucher and Badri recorded the predictive performance of metric values at weight ratios 5%, 10%, 15%, and up to 95% and their results indicated the better outcome at 30%, consequently, they chose the metric value at this ratio as the final cut-off point. On the contrary, the results of our study produced the different outcome, as in some cases, the metric value at 20% were better separating the faulty from non-faulty classes, whereas in other cases, the metric values at 30% were showing the best results. Figure 2 shows, for WMC metric, the predictive results of Alves Rankings at weight ratios 20% and 30% of all investigated systems. In case of Ant-1.7, the FPR, FNR, and G-mean values of the threshold at 20% are 0.331, 0.216, and 0.729 respectively, whereas, the same values in the case of threshold at 30% are 0.193, 0.364, and 0.714 respectively. The outcome indicates the slightly better degree of fault prediction in case of threshold at weight ratio 20%. Similarly, the observation revealed the best predictive results of thresholds at weight ratio 20% for the software systems- Ivy, Lucene-2.4, Poi-3.0, Tomcat, Velocity, Xerces, and Xalan, whereas, in the remaining systems the best predictive outcome was received at ratio 30%. Similarly, this variation was also observed for other metrics. In this scenario, there exist three options to select the appropriate weight ratio.

—First to select the metric value at Alves 20% as a threshold.

—Second to opt metric value at Alves 30% as a threshold.

—And the last is to select the threshold best from both of the weight ratio.

In this paper, the third option has been opted for. Based on the adopted approach, for WMC metric, the metric values corresponding to the weight ratio 30% were selected as the thresholds in the software systems- Arc, Camel-1.6, Log4j-1.2, Prop-6, Redactor and Synapse. Whereas, for the remaining software systems, the metric values at 20% were considered as the threshold. And the results yielded from these selected thresholds were further used for comparison purpose in the upcoming subsection.
Figure 3 shows the performance achieved by the threshold values identified through ROC and

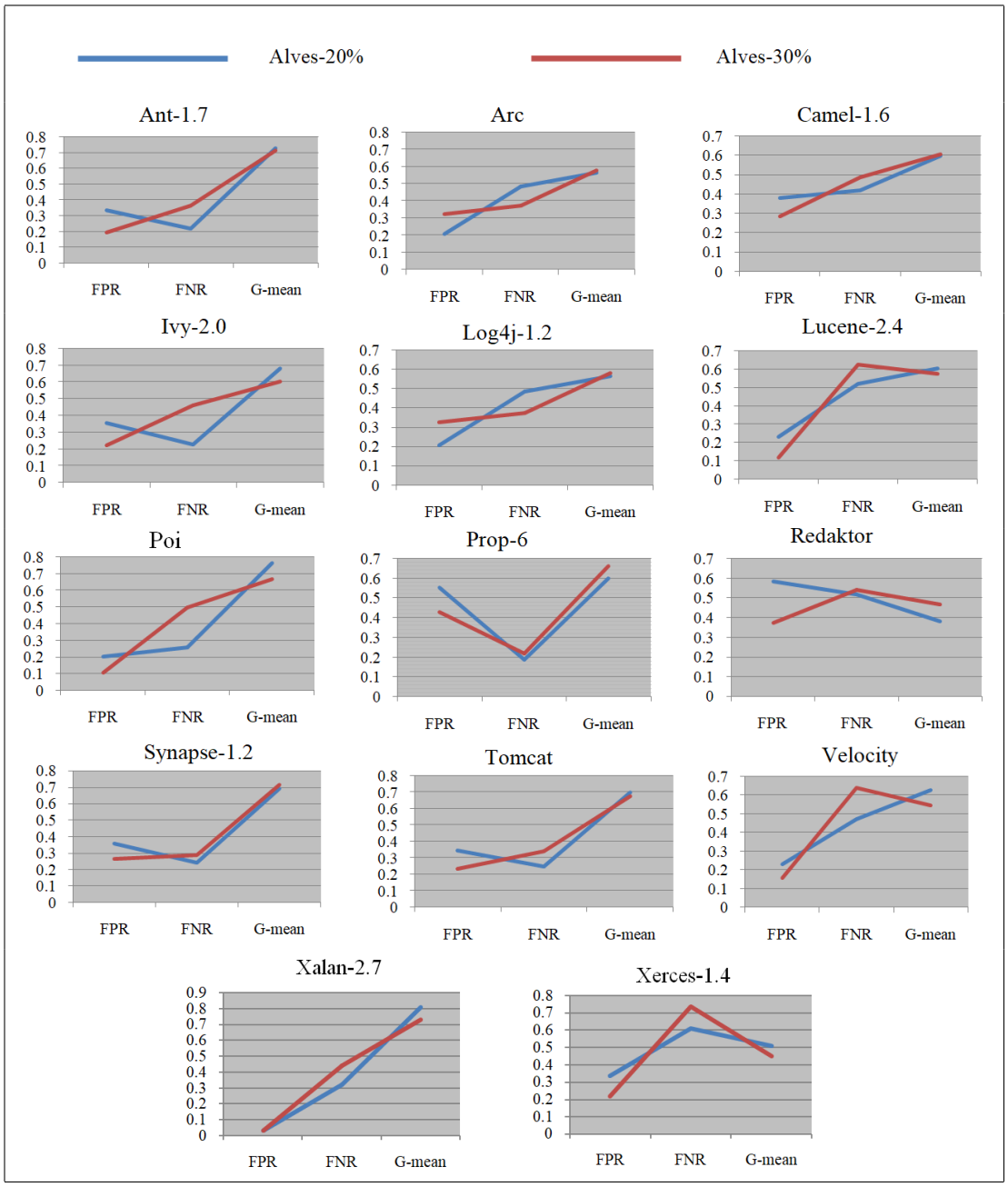

Figure 2. Prediction results at Alves Rankings 20% and Alves Rankings 30%

Alves Rankings techniques. Due to space constraint, the results of only first five software systems are depicted through bar charts. In case of software system Ant, the observations revealed that threshold obtained for RFC showed higher association with binary outcome as they came up with the lowest number of false positives and false negatives and higher G-mean value. Moreover, the acceptable results were obtained from the thresholds of LOC, WMC, NPM, and CAM metrics, whereas CBO, Ce, MOA, AMC, and Max_CC gave average results. In case of DAM, the thresholds derived through both techniques showed very high false positive rate. For metric Ce, NPM, and Max_CC, the thresholds retrieved through ROC and Alves Rankings are same. Whereas for

CAM, threshold obtained through the ROC curve produced far better predictive capability than Alves Rankings, as in the latter case the false negative rate is very high.

In Arc software system, the threshold values obtained through both ROC and Alves techniques showed overall poor results. The DAM and Max_CC thresholds showed the worst predictive performance. Only the threshold of Ce metric identified through the ROC curve showed the acceptable predictive results. The thresholds of CBO and CAM measures showed the average predictive capability of identifying faulty classes.
In Ivy software system, the findings exhibit the acceptable results of the thresholds of WMC, LOC and RFC metrics identified through ROC curve. On the other hand, for MOA metric, ROC curve was found inefficient to identify the appropriate cut-off point which can proficiently discriminate the classes into either faulty or non-faulty groups. The thresholds identified for DAM showed very high false positive rate. In Log4j, the metric thresholds of LOC, CAM, AMC, and Max_CC identified through the ROC curve showed very high false positive rate. The values of RFC and MOA brought forth the worst predictive outcome.
In Xalan software system, RFC metric turned up as the best predictor for determining the faulty classes, however, the excellent results were also observed for thresholds identified for WMC, CBO, LOC, and NPM measures. The thresholds retrieved for CAM metric through Alves Rankings showed very high false positive rate. Overall the threshold yielded by ROC curve gave superior results than Alves Rankings technique.
In software system Xerces, the thresholds computed for RFC and AMC through Alves Rankings showed very high false negative rate. Contrarily, the metric thresholds of CBO and Ce determined through the ROC curve exhibited the excellent predictive capability. Both techniques failed to identify an optimal threshold for the MOA measure, as the false negative rate of the identified values is very high. The metric thresholds of ROC curve produced overall better results as compared to the Alves Rankings technique.
In software system Redactor, only the thresholds identified for Ce and AMC produced the average results, whereas remaining metrics threshold showed very poor predictive capability. The detailed results about the classification capability possessed by ROC curve and Alves Rankings in each of the software systems are shown in Table IX and Table X respectively. In the table, I column contains the list of measures which showed poor classification ability, whereas, II, III, IV, V, and VI contains the list of measures which showed poor, average, acceptable, excellent, and outstanding capabilities in discriminating the faulty and non-faulty classes.

Important findings elicited from the analysis of the experimental results are summarized below:

—In case of Redactor, the outcome revealed the failure of both techniques in identifying the appropriate metric thresholds. The resultant performance indices showed the insignificant capability of the identified thresholds in putting the classes in the right category.

—In case of Xalan, the threshold computed by the ROC curve produced the overall better predictive results as compared to other software systems. The thresholds identified for the metrics produced the lowest number of incorrect predictions and as a result highest G-mean values as compared to other systems.

—In Xerces, the threshold values calculated through the ROC curve gave significantly better predictive outcome than the values obtained through the Alves Rankings.

—In Software tomcat, the threshold values of AMC and Max_CC identified through Alves Rankings showed poor predictability in detecting the faulty classes. Whereas, the results of the same metrics calculated through the ROC curve showed a significantly better ability to discriminate between faulty and non-faulty classes.

—In the software systems- Arc, Poi, Xalan, and Xerces, the threshold values of Ce (identified through ROC curve) showed significant ability to separate the faulty and non-faulty classes.

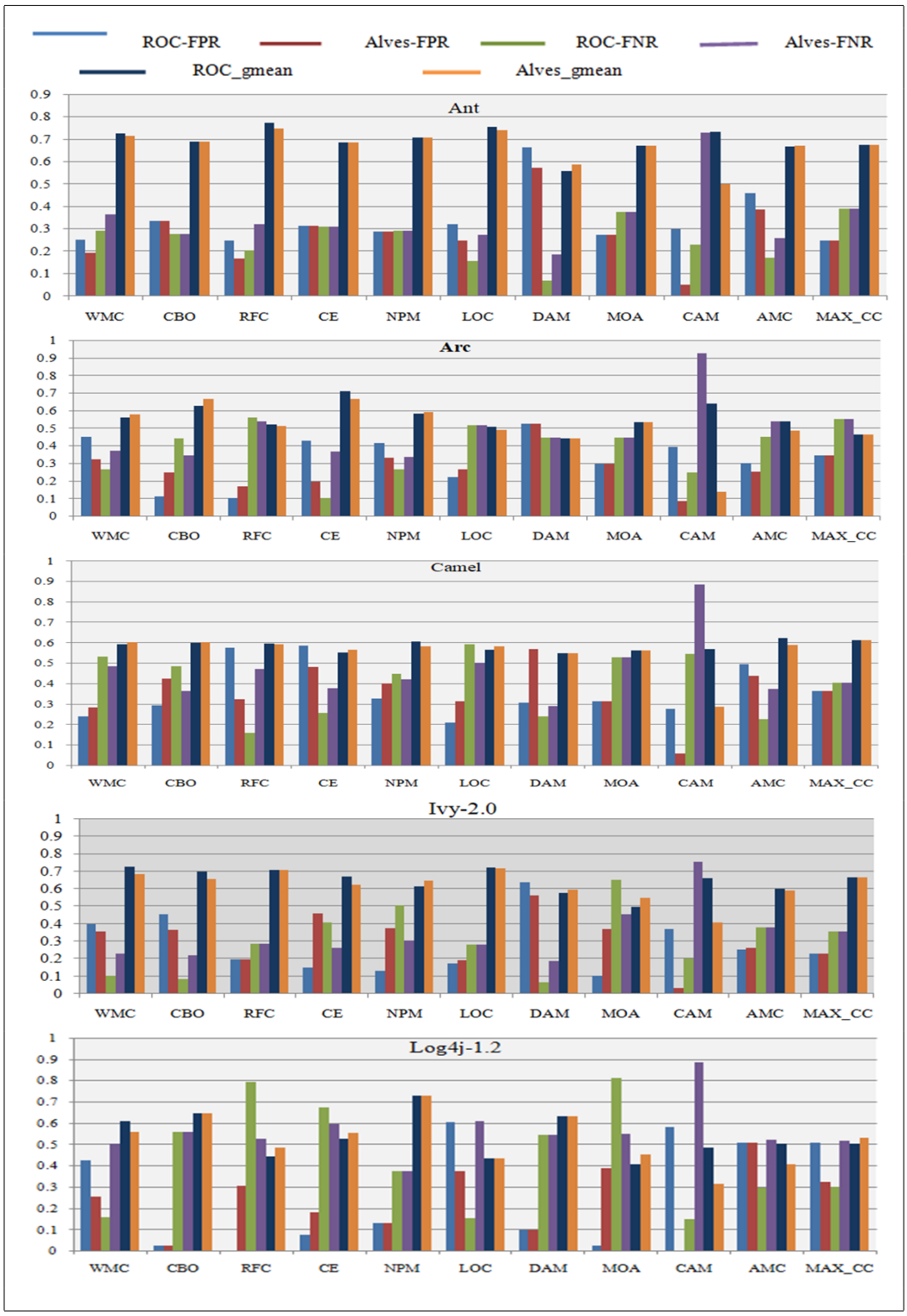

Figure 3. Prediction results based on ROC curve and Alves Ranking techniques

Software	I	II	III	IV	V	VI
Ant	*	DAM	CBO, Ce, MOA, AMC, Max_CC	WMC, RFC, NPM, LOC, CAM	*	*
Arc	DAM, Max_CC	WMC, RFC, NPM, LOC, MOA, AMC	CBO, CAM	Ce	*	*
Camel-1.6	*	WMC, CBO, RFC, Ce, LOC, DAM, MOA, CAM	NPM, AMC, Max_CC	*	*	*
Ivy-2.0	MOA	DAM, AMC	CBO, Ce, NPM, CAM, Max_CC	WMC, RFC, LOC	*	*
Log4j-1.2	LOC, RFC, MOA, CAM	Ce, AMC, Max_CC	WMC,CBO, DAM	NPM	*	*
Lucene-2.4	*	MOA, AMC, Max_CC	WMC, CBO, RFC, Ce, NPM, LOC, DAM, CAM	*	*	*
Poi-3.0	*	MOA	CAM, AMC	WMC, RFC, CBO, Ce, NPM, LOC, DAM, Max_CC	*	*
Prop-6	*	Ce, DAM, AMC	CBO, RFC, CAM, MOA, Max_CC	WMC, NPM, LOC	*	*
Redaktor	WMC, CBO, DAM, MOA, Max_CC	RFC, NPM, LOC, CAM	Ce, AMC	*	*	*
Synapse-1.2	*	CBO, Ce, NPM	DAM, MOA, AMC, Max_CC	WMC, RFC, CAM, LOC	*	*
Tomcat	*	NPM	WMC, DAM, Max_CC	CBO, RFC, CAM, MOA, AMC	LOC	*
Velocity-1.6	Max_CC	MOA, AMC	WMC, CBO, LOC, RFC, Ce, NPM, DAM	CAM	*	*
Xalan-2.7	MOA, CAM	Max_CC	DAM	Ce, AMC	WMC, LOC, CBO, RFC, NPM	*
Xeres-1.4	*	WMC, DAM, MOA	RFC, CAM, Max_CC	NPM, AMC, LOC	CBO, Ce,	*

Table IX: The classification ability of the thresholds identified through ROC curve

—The cut-off value obtained for RFC showed a higher degree of fault predictability in all of the software systems, except the Log4j and Redactor.

—The observations based on the outcome of performance measures revealed that after RFC the threshold values retrieved for WMC, LOC, CBO, and NPM measures produced the best predictive efficiency.

From the above findings, it can be concluded that CK metrics, such as, RFC, WMC, LOC, and CBO are the best predictor of fault-proneness. Therefore, the results of this work support the findings of the study [14], where the authors found the significant predictive efficiency of the CK metric thresholds with fault-proneness. Besides the previously mentioned metrics, the current study also found other measures, such as CAM, Ce and NPM, as an efficient predictor of fault-proneness. These measures gave acceptable results in the majority of the software systems. The coupling and size measure appeared at the top of the best predictor list. Whereas, metrics like DAM, MOA, AMC, and Max_CC showed the average capability in discriminating the faulty

Software	I	II	III	IV	V	VI
Ant	CAM	DAM	CBO, Ce, MOA, AMC, Max_CC	WMC, RFC, NPM, LOC	*	*
Arc	LOC, DAM, CAM, AMC, Max_CC	WMC, RFC, NPM, MOA	CBO, Ce	*	*	*
Camel-1.6	CAM	RFC, Ce, NPM, LOC, DAM, MOA, AMC	WMC, CBO, Max_CC	*	*	*
Ivy-2.0	CAM	DAM, MOA, AMC	WMC, CBO, Ce, NPM, Max_CC	RFC, LOC	*	*
Log4j-1.2	RFC, LOC, MOA, CAM, AMC	WMC, Ce, Max_CC	CBO, DAM	NPM	*	*
Lucene-2.4	CAM	LOC, MOA, AMC, Max_CC	WMC, CBO, RFC, Ce, NPM, DAM	*	*	*
Poi-3.0	CAM	MOA, Max_CC	RFC, LOC, DAM, AMC	WMC, CBO, Ce, NPM	*	*
Prop-6	*	Ce NPM, CAM, DAM, AMC	WMC, CBO, RFC, LOC, MOA, Max_CC	*	*	*
Redaktor	WMC, CBO, RFC, NPM, DAM, MOA, CAM, Max_CC	LOC	Ce, AMC	*	*	*
Synapse-1.2	*	CBO, Ce, NPM	DAM, MOA, AMC, LOC, Max_CC	WMC, RFC, CAM	*	*
Tomcat	AMC, CAM, Max_CC	NPM	WMC, DAM	CBO, RFC, MOA	LOC	*
Velocity-1.6	AMC, CAM, Max_CC	CBO, LOC, NPM, MOA	WMC, RFC, Ce, DAM	*	*	*
Xalan-2.7	MOA, CAM	CBO, Ce, LOC	DAM, AMC, Max_CC	RFC	WMC, NPM	*
Xeres-1.4	RFC, CAM	WMC, NPM, DAM, MOA, AMC, LOC, Max_CC	CBO	*	Ce	*

Table X: The classification capability of the thresholds identified through Alves Rankings

and non-faulty classes.

4.4 Comparison of threshold techniques

During the experiment, Wilcoxon signed rank test was applied to perform the comparison between the predictive performance of ROC and Alves Rankings and to identify the best algorithm between them. In the study Demsar [2006], the author proved the conceptual unsuitability of t-test in comparing the classifiers performance and showed the effectiveness of the Wilcoxon test for the purpose. The null and alternate hypotheses of this test are as follows.
Null hypothesis: Both ROC curve and Alves Rankings shares same predictive capability.
Alternate hypothesis: The predictive efficiency of ROC curve and Alves Rankings is not same.
As discussed earlier, for the considered methods, the positive and negative ranks are extracted from the computed values of the performance measures. Then, the difference between both ranks are analyzed and if its p-value is less than .05 only then the difference is considered as statistically significant and in this case the null hypothesis will be rejected, otherwise not. In this paper, the Wilcoxon test is applied separately on the performance outcome of each of the metrics selected so

far. In order to execute the test, we make a use of very well-known statistical package, i.e. SPSS. Table XI contains the results based on the Wilcoxon test, where, asterisk sign (*) represents the insignificant difference between the predictive abilities of ROC and Alves Rankings as here the p-value is greater than 0.05. The cases, where ROC methods predictability is statistically better Alves Rankings are represented by the resultant p-value, followed by (R) symbol. The important findings withdrawn on the application of Wilcoxon test are as follows:

—In case of WMC, CBO, Ce, DAM, MOA, and Max_CC, no significant difference was found between the performance of ROC curve and Alves Rankings.

—In case of RFC, NPM, LOC, CAM, and AMC, the results revealed the better predictive performance of ROC curve as compared to Alves method.

From the findings, we can conclude that ROC curve performed significantly better than the Alves Rankings technique.

	WMC	**CBO**	**RFC**	**Ce**	**NPM**	**LOC**	**DAM**	**MOA**	**CAM**	**AMC**	**Max_CC**
P-value	*	*	.011(R)	*	.029(R)	.003(R)	*	*	<.000(R)	.016	*

Table XI: Results based on Wilcoxon signed rank test

5. RESULTS BASED ON CROSS-VERSIONS FAULT PREDICTION

This section contains the findings of the experiment conducted to validate the idea of cross-version fault prediction. Here, the threshold values are calculated from one version, and the computed values are assessed for their discrimination ability in the subsequent versions of the same software system. Boucher and Badri [2017] also performed the cross-version fault-prediction and selected Ant software with versions 1.3, 1.4, 1.5, 1.6, and 1.7, to carry out the experiment. Their study implemented the experiment in two ways. First, the values obtained from the threshold model of the previous version were validated in the model of its subsequent version. Second, the model was constructed from all of the previous versions and obtained value was tested on the subsequent version. Acceptable results were obtained from the experiment.

In this study, we executed the cross-version experiment on the releases of the JEdit software system, i.e., 4.2 and 4.3. As mentioned earlier, the dataset utilized to implement the idea of cross-version fault prediction is collected from the public repository. Among the available software systems, we only picked the metric details of JEdit, as other software systems were containing the information about the method-level metrics and the current study is concentrating only on the OO metrics. The training process was conducted on the JEdit-4.2, and the derived thresholds were applied to the subsequent version, JEdit-4.3. It is necessary to mention here that the accessed repository contains the metric data of only 7 metrics- WMC, DIT, NOC, CBO, RFC, LCOM, and NPM. Therefore, the experiment was conducted only on the available software metric. Table XII shows the threshold values retrieved after the application of ROC curve and Alves Rankings for JEdit-4.2 software system. In case of Alves method, the metric values at weight ratio 30% were showing the best results, therefore, the metric values corresponding to 30% were chosen for the experiment.

Figure 4 depicts the predictive performance of the threshold values of JEdit-4.2 when applied

Method	**WMC**	**DIT**	**NOC**	**CBO**	**RFC**	**LCOM**	**NPM**	**LOC**
ROC curev	6	3	1	7	42	62	6	100
Alves	4	1	0	7	21	37	3	80

Table XII: Threshold values calculated for JEdit- 4.2

on JEdit-4.3. The outcome showed the failure of DIT and NOC, in separating the classes based on their fault status. These metrics were also found insignificant in the last subsection for other software systems. The thresholds derived by using ROC curve for WMC metric shows average classification ability, whereas, the threshold values of RFC metric shows poor classification ability. In case of CBO, both techniques gave fair results. The software metrics, such as LOC, NPM, and RFC which were identified as the best predictors in the last section, shows poor predictive efficiency in case of cross-version experiment. On the comparison of ROC and Alves Rankings, the former method exhibited slightly better performance.

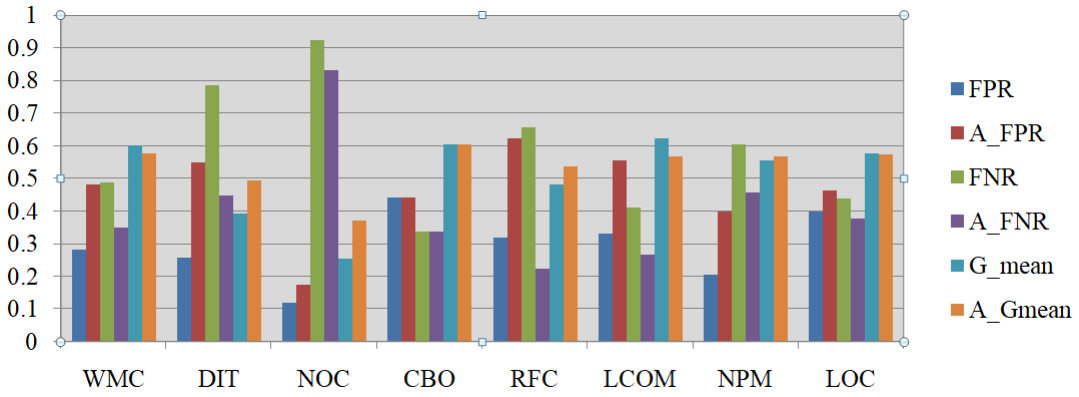

Figure 4. Predictive results of cross-version experiment

Based on the prediction outcome, it can be concluded that the idea of cross-version threshold application failed to give appropriate results for JEdit software system. On the contrary, the outcome of the study Boucher and Badri [2017] showed better prediction results when the thresholds were shared between the releases of the Ant software system. Therefore, in order to confirm the results, more tests need to be conducted on other software systems.

6. RESULTS BASED ON CROSS-PROJECT FAULT PREDICTION

The current study has also assessed the validity of cross-project fault prediction by testing the applicability of the thresholds, derived from one software system, on different software systems. Malhotra and Bansal [2015] executed this idea by sharing the thresholds among the software systems having similar characteristics. The authors identified the thresholds of Ivy and tested them on Ant and Tomcat. The common features possessed by these software systems are a) all of them are written in Java, b) they are open source software and released under Apache software license i.e. software belongs to the same organization. Similarly, JEdit and Sakura were chosen on the grounds of similar software characteristics: both of them are text editors and are of similar size. Boucher and Badri [2017] also based their software selection strategy on the characteristics similarity approach and backed the cross-project experiment with the software systems belonging to the same organization and having similar size. Besides this, they also tested the validity of cross-project fault prediction on the software systems belonging to distinct organizations, but are of similar size.

The current study also executed the cross-project fault prediction, but applied different criteria for the software selection. The selection of software systems was based on their similarity in terms of coding characteristics. In contrary to the decisive factor chosen in study Boucher and Badri [2017], where the software systems having similar external characteristics were chosen for the analysis, the current study performs the same experiment by selecting the software systems

possessing similar coding features. The two conditions which are required to be fulfilled by the software system for inheriting the thresholds of another system are:

1) The number of the classes in both software systems should be approximately same.
2) The size of the classes in both software systems should follow an approximately similar pattern.

Among the available software systems, Arc, Synapse, and Velocity were put under the same category, as all of them satisfied both of the above conditions. Similarly, software systems Lucene and Ivy were put into the same category. Only those measures were considered for the experiment, which were proved as an efficient predictor in the Section 4. Thus, the selected measures were WMC, CBO, RFC, Ce, NPM, CAM and LOC. Also, for the experiment, we considered the technique which gave the best results in Section 4, i.e. ROC curve.
Figure 5 exhibits the outcome when we used the thresholds of Arc software on Synapse and Velocity. The figure (a) contains the performance results when the thresholds of Arc were applied on Synapse. Acceptable results are obtained in case of RFC and CAM, whereas, WMC and LOC yielded average results. And, (b) shows the results obtained when the thresholds of Arc system were applied on Velocity. Overall, average results are obtained for all metrics, but revealed low predictive efficiency when compared to the results of Figure 5(a). It is necessary to mention here that when the thresholds of Arc software were validated on its own testing dataset, then the overall average performance was observed in separating the faulty and non-faulty classes. In fact, the predictive results of Synapse and Velocity, on applying Arcs thresholds are considerably better than the results obtained by the Arc itself. Figure 6 shows the fault prediction results when the threshold values of Ivy were applied on the Lucene system. The metrics WMC, CBO, CAM, and Ce revealed average classification ability.
Furthermore, In order to get more insight into the results of the cross-project fault prediction,

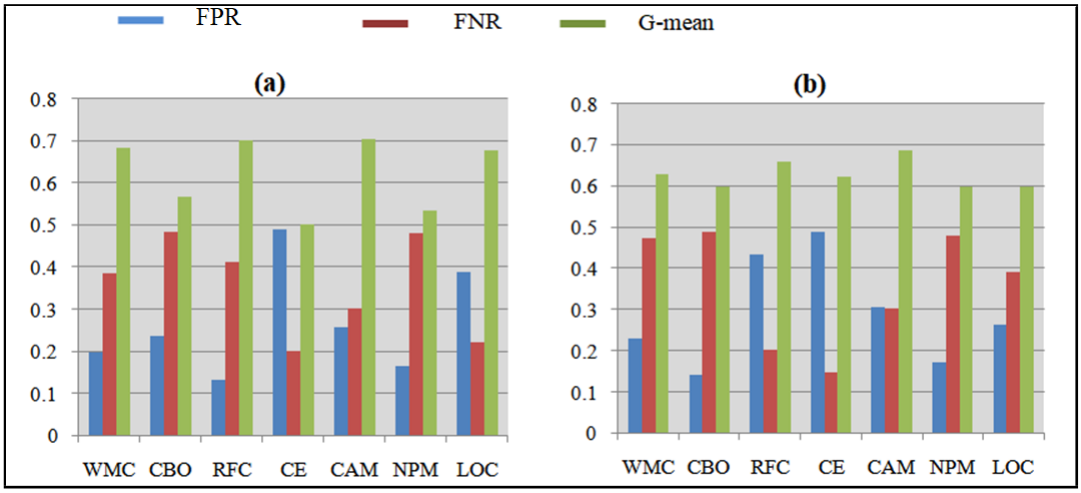

Figure 5. Results based on the application of Arcs threshold on the software system Synapse (a) and Velocity (b)

we compared the predictive performance of the thresholds when applied on the different system with the performance when applied to the training subset of the same software. Figure 7 (a) contains the G-mean value of software Synapse when using its own thresholds and when using the threshold values of Arc. The results showed almost similar predictive efficiency of thresholds on both software systems. Figure 7 (b) contains the G-mean values of software system Velocity when using its own thresholds and when inheriting the threshold of Arc. Similarly, Figure 8 contains the G-mean values obtained by software Lucene when using its own threshold and the G-mean obtained by Lucene when inherited the thresholds of Ivy.
Based on the above findings, it can be concluded that by selecting the software systems fulfilling

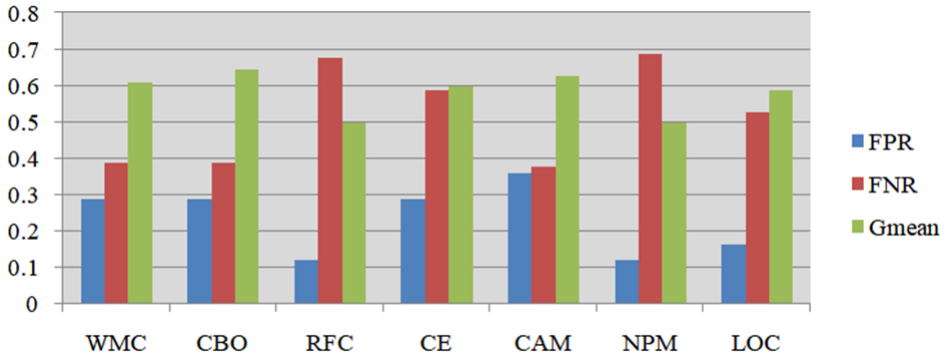

Figure 6. Results based on the application of Ivy threshold on Lucene

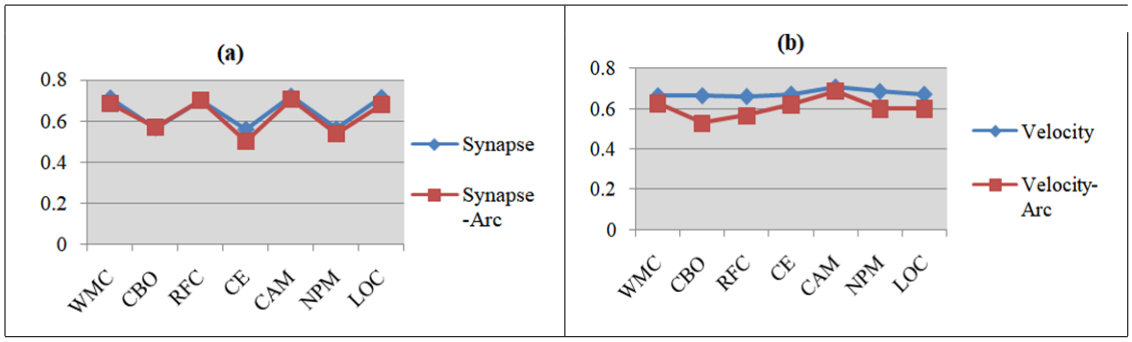

Figure 7. Comparison between the predictive results of cross-validation and cross-project fault prediction.

the predefined conditions for the experiment of cross-project fault prediction, acceptable and average classification results can be obtained.

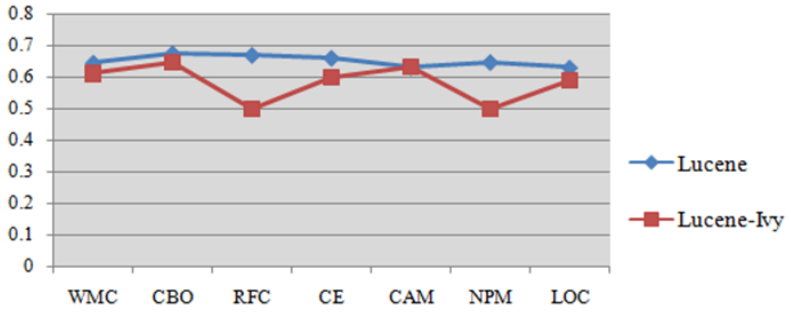

Figure 8. Comparison between the predictive results of cross-validation and cross- project fault prediction.

7. THREATS TO VALIDITY

For the current study, the dataset size is large enough to extract meaningful conclusions. However, the findings of the study cannot be generalized to other software systems.

Although, cross-project experiment gave acceptable and average results, but their applicability on other software is still questionable. However, this can be reduced by replicating the approach across a wider variety of systems in the future work. And, for cross-version validation, the

experiment is performed on one software system, so the finding withdrawn from them cannot be generalized to other systems.
In the case of Alves ranking, we selected the metric values either at weight ratio 20% or 30% (whichever was giving the best results) as the cut-off point. The outcome might have been different if a fixed weight ratio was chosen. This threat can be reduced by replicating the study choosing the metric values corresponding to one weight ratio.

8. CONCLUSIONS AND FUTURE WORK

In the current paper, the authors have focused on the identification of optimal metric thresholds which can proficiently separate the faulty classes from non-faulty classes. This prognostic capability can help the testers in the careful apportionment of the testing efforts for their systematic utilization. The metric thresholds calculated through ROC curve and Alves Rankings were assessed for their fault prediction efficiency. This study also compared the considered threshold techniques for their abilities to determine the optimal metric thresholds. Some of the important findings from the study are summarized as follows:

—On the application of ULR, out of 20 selected OO metrics only WMC, CBO, RFC, Ce, NPM, LOC, DAM, MOA, MFA, CAM, AMC, and Max_CC, showed the statistically significant relationship with the fault-proneness. So, these 12 metrics were selected for the next step of threshold calculation.

—The RFC, CBO, WMC, Ce, CAM, and NPM emerged as the leading efficient measures in discriminating the faulty and non-faulty classes, whereas DAM, MOA, AMC, and Max_CC showed moderate ability in discriminating the fault-prone from the fault-free classes.

—The threshold values of CAM metric (in all software systems) identified through ROC showed the supremacy over Alves Rankings in predicting the faulty classes.

—Both threshold techniques produced the best metric thresholds for software Xalan, as the derived threshold values successfully classified the classes into faulty and non-faulty category with lowest false negatives and false positives and highest G-mean value. On the contrary, the threshold calculated for software Redactor showed the opposite results by yielding poor predictive capability.

—During the comparison of considered techniques through Wilcoxon signed rank test, the finding revealed that there exists the significant difference between the predictive performances of the ROC curve and Alves Rankings techniques. Furthermore, by analyzing the significant differences between the positive and negative ranks, we found that ROC curve worked better than Alves Rankings. The ROC curve yielded better results in case of RFC, NPM, LOC, CAM, and AMC.

—In the cross-version experiment, WMC and CBO showed average results. And, Alves Ranking gave slightly better results than ROC curve.

—In the cross-project experiment, the acceptable results were procured on the application of arc thresholds on the synapse, whereas on applying Ivys threshold on Lucene, overall average results were obtained.

Based on the findings, we can conclude that the probability of getting good results through the ROC curve is high; therefore researchers and practitioners can use them in the field of fault prediction for the purpose of further research and can also be utilized in industrial environment. In future work, we will replicate the study by assessing more threshold techniques. Also, in this study, an assessment of the threshold techniques was conducted for the binary classification. Future work can be conducted to investigate the threshold techniques for predicting the number of bugs. Also, more tests can be done in the cross-version and cross-project fault prediction.

References

Aggarwal, K., Singh, Y., and Malhotra, R. 2007. Investigating effect of design metrics on fault proneness in object-oriented systems. *Journal of Object Technology Vol.6,* No.10, pp.127–141.

Alves, T. L., Ypma, C., and Visser, J. 2010. Deriving metric thresholds from benchmark data. In *IEEE International Conference on Software Maintenance.*

A.M.Ferreira, A.S.Bigonha, M., S.Bigonha, R., F.O.Mendes, L., and C.Almeida, H. 2012. Identifying thresholds for object-oriented software metrics. *The Journal of Systems and Software Vol.85,* No.2, pp.244–257.

Arar, O. F. and Ayan, K. 2016. Deriving thresholds of software metrics to predict faults on open source software: Replicated case studies. *Expert Systems with Applications Vol.61*, pp.106–121.

Bansiya, J. and Davis, C. 2002. A hierarchical model for object-oriented design quality assessment. *IEEE Transactions on Software Engineering Vol.28,* No.1, pp.4–17.

Basili, V., Briand, L., and Melo, W. 1996. A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering Vol.22,* No.10, pp.751–761.

Bender, R. 1999. Quantitative risk assessment in epidemiological studies investigating threshold effects. *Biometrical Journal, Vol.41,* No.3, pp.305–319.

Boucher, A. and Badri, M. 2017. Software metrics thresholds calculation techniques to predict fault-proneness: An empirical comparison. *Information and Software Technology Vol.96*, pp.38–67.

Chidamber, S. and Kemerer, C. 1994. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering Vol.20,* No.6, pp.476–493.

Demsar, J. 2006. Statistical comparisons of classifiers over multiple data sets. *Biometrical Journal Vol.7,* No.1, pp.1–30.

Elish, M. O. 2012. A comparative study of fault density prediction in aspect-oriented systems using mlp, rbf, knn, rt, denfis and svr models. *Artificial Intelligence Review Vol.42,* No.4, pp.695703.

Erdreich, L. S. and Lee, E. T. 1981. Use of relative operating characteristic analysis in epidemiology: A method for dealing with subjective judgement. *American Journal of Epidemiology Vol.114,* No.5, pp.649–662.

Erni, K. and Lewerentz, C. 1996. Applying design-metrics to object-oriented frameworks.

French, V. 1999. Establishing software metric thresholds.

Halstead, M. 1977. Elements of software science. Elsevier Science.

Henderson-Sellers, B. 1996. Software metrics. WG Sales.

Jiang, Y., Cukic, B., and Ma, Y. 2008. Techniques for evaluating fault prediction models. *Empirical Software Engineering Vol.13,* No.5, pp.561595.

Kaur, A. and Malhotra, R. 2008. Application of random forest in predicting fault-prone classes.

Khoshgoftaar, T., Gao, K., and Szabo, R. 2001. An application of zero-inflated poisson regression for software fault prediction.

Kubat, M. and Matwin, S. 1997. Addressing the curse of imbalanced training sets: one-sided selection. *Icml Vol. 97*, pp.179–186.

Kumar, L., Misra, S., and Rath, S. K. 2017. An empirical analysis of the effectiveness of software metrics and fault prediction model for identifying faulty classes. *Computer Standards and Interface Vol.53*, pp.1–32.

lLov Kumar, Sripada, S. K., Sureka, A., and Rath, S. K. 2018. Effective fault prediction model developed using least square support vector machine (lssvm). *Journal of Systems and Software Vol.137*, pp.686712.

Mahajan, R., KumarGupta, S., and Bedi, R. K. 2015. Design of software fault prediction model using br technique. *Procedia Computer Science Vol.46*, pp.849–858.

Malhotra, R. and Bansal, A. J. 2015. Fault prediction considering threshold effects of object-oriented metrics. *Expert Systems Vol.32,* No.2, pp.203–219.

Malhotra, R. and Jain, A. 2012. Fault prediction using statistical and machine learning methods for improving software quality. *Journal of Information Processing Systems Vol.8,* No.2, pp.241–262.

Malhotra, R., Kaur, A., and Singh, Y. 2011. Empirical validation of object-oriented metrics for predicting fault proneness at different severity levels using support vector machines. *International Journal of System Assurance Engineering and Management Vol.1,* No.3, pp.269–281.

Malhotra, R. and Singh, Y. 2011. On the applicability of machine learning techniques for object oriented software fault prediction. *Software Engineering: An International Journal Vol.1,* No.1, pp.24–27.

Martin., R. 1994. Object-oriented design quality metrics an analysis of dependencies. Object-Oriented, Proceedings of Workshop Pragmatic and Theoretical Directions in Software Metrics.

McCabe, T. J. 1976. A complexity measure. *IEEE Transactions on Software Engineering* No.4, pp.308–320.

Nejmeh, B. A. 1988. Npath a measure of execution path complexity and its applications. *Communications of the ACM Vol.31,* No.2, pp.188–200.

Ostrand, T., Weyuker, E., and Bell, R. 2005. Predicting the location and number of faults in large software systems. *IEEE Transactions on Software Engineering Vol.31,* No.4, pp.340 – 355.

Radjenovi, D., Heriko, M., Torkar, R., and ivkovi, A. 2013. Software fault prediction metrics: A systematic literature review. *Information and Software Technology Vol.55,* No.8, pp.1397–1418.

Rajaraman, C. and Lyu, M. R. 1992. Reliability and maintainability related software coupling metrics in c++ programs. In *Proceedings International Symposium on Software Reliability Engineering (ISSRE).*

Rosenberg, L. H. and Hyatt, L. E. 1997. Software quality metrics for object-oriented environments. *Crosstalk Journal Vol.10,* No.4, pp.1–6.

Shatnawi, R. 2010. A quantitative investigation of the acceptable risk levels of object-oriented metrics in open-source systems. *IEEE Transactions on Software Engineering Vol.36,* No.2, pp.216–225.

Shatnawi, R. 2015. Deriving metrics thresholds using log transformation. *Journal of Software: Evolution and Process Vol.27,* No.2, pp.95–113.

Shatnawi, R., Li, W., Swain, J., and Newmani, T. 2010. Finding software metrics threshold values using roc curves. *Journal of software maintenance and evolution: Research and practice Vol.22,* No.1, pp.1–16.

Singh, S. and Kahlon, K. S. 2014. Object oriented software metrics threshold values at quantitative acceptable risk level. *CSI Transactions on ICT Vol.2*, pp.191–205.

Tang, M.-H., Kao, M.-H., and Chen, M.-H. 1999. An empirical study on object-oriented metrics.

Thwin, M. M. T. and Quah, T.-S. 2002. Application of neural network for predicting software development faults using object-oriented design metrics.

Van Koten, C. and Gray, A. 2006. An application of bayesian network for predicting object-oriented software maintainability. *Information and Software Technology Vol.48,* No.1, pp.59–67.

Youden, W. J. 1950. Index for rating diagnostic tests. *Cancer Vol.3,* No.1, pp.32–35.

Zar, J. H. 1999. Biostatistical analysis. Pearson Education India.

Navneet Kaur is a Ph.D. student at the Department of Computer Science, Guru Nanak Dev University, Amritsar. Her areas of interest are Software Quality, Software metrics, and Object-Oriented programming.

Dr. Hardeep Singhi is a Professor and Head at the Department of Computer Science, Guru Nanak Dev University, Amritsar, India. His research interests lie within Software Engineering and Information Systems.