Fault Prediction Using Statistical and Machine Learning Methods for Improving Software Quality

Ruchika Malhotra* and Ankita Jain**

Abstract—An understanding of quality attributes is relevant for the software organization to deliver high software reliability. An empirical assessment of metrics to predict the quality attributes is essential in order to gain insight about the quality of software in the early phases of software development and to ensure corrective actions. In this paper, we predict a model to estimate fault proneness using Object Oriented CK metrics and QMOOD metrics. We apply one statistical method and six machine learning methods to predict the models. The proposed models are validated using dataset collected from Open Source software. The results are analyzed using Area Under the Curve (AUC) obtained from Receiver Operating Characteristics (ROC) analysis. The results show that the model predicted using the random forest and bagging methods outperformed all the other models. Hence, based on these results it is reasonable to claim that quality models have a significant relevance with Object Oriented metrics and that machine learning methods have a comparable performance with statistical methods

Keywords—Empirical Validation, Object Oriented, Receiver Operating Characteristics, Statistical Methods, Machine Learning, Fault Prediction

1. INTRODUCTION

Software reliability is a critical field in software engineering and an important facet of software quality. Every organization wants to assess the quality of the software product as early as possible so that poor software design leading to lower quality product can be detected and hence be improved or redesigned. This would lead to significant savings in the development costs, decrease the development time, and make the software more reliable. The quality of the software can be measured in terms of various attributes such as fault proneness, maintenance effort, testing effort, etc. In this study, we have used fault proneness as the quality predictor. Fault proneness is defined as the probability of fault detection in a class [1-4]. Due to high complexity and constraints involved in the software development process, it is difficult to develop and produce software without faults. High cost is involved in finding and correcting faults in software projects. Thus, we need to identify or locate the areas where more attention is needed in order to find as many faults as possible within a specified time and budget. To address this issue, we predict fault proneness model using statistical and machine learning methods in this paper. One of the approaches to identify faulty classes early in the development cycle is to predict models by using software metrics. In the realm of an object oriented environment, object oriented soft-

Manuscript received May 16, 2011; first revision December 22, 2011; accepted February 13, 2012.

Corresponding Author: Ruchika Malhotra
* Dept. of Software Engineering, Delhi Technological University, Delhi, India (ruchikamalhotra2004@yahoo.com)
** Dept. of Computer Engineering, Delhi Technological University, Delhi, India (ankita4813@yahoo.com)

ware metrics have become increasingly popular with researchers. There are various object-oriented metrics available in the literature [5-11] to predict software quality attributes.

Hence, the main contributions of this paper are: (1) To establish relationship between object oriented metrics and fault proneness. There are a number of object oriented metrics such as CK metrics [5], MOOD [7], QMOOD metrics [8], etc., but not all the metrics are good predictors of fault proneness. Thus, it is very important to understand the relationship of object oriented metrics and fault proneness. In other words, we must find out which of the metrics are significant in predicting the faulty classes. Then, these significant metrics can be combined into one set to build the multivariate prediction models for predicting fault proneness. Identified metrics will help software practitioners to focus on fault prone classes and ensure a higher quality software product with the available resources. Software researchers may use these metrics in further studies. (2) To analyze machine learning methods (method of programming computers to optimize performance criterion using example data or past experience). Nowadays, machine learning is widely used in various domains (i.e., retail companies, financial institutions, bioinformatics, etc.) There are various machine learning methods available. We have used six machine learning methods to predict the accuracy of the model predicted. These six machine learning methods have been widely used in literature and have shown good results [4, 12-14]. Amongst the various models predicted, we must determine one of the models to be the best model, which can be used by researchers in further studies to predict the faulty classes.

In order to achieve this aim we have used dataset collected from open source software, poi [15]. This software was developed using Java language and consists of 422 classes. The different dataset used by us will provide an important insight to researchers for identifying the relevance of metrics with a given type of dataset. Since it is Open Source software, the users have freedom to study and modify the source code (written in Java) without paying royalties to previous developers. We have used one statistical method (logistic regression) and six machine learning methods (random forest, adaboost, bagging, multilayer perceptron, support vector machine, and genetic programming).

We have analyzed the performance of the models by calculating area under the Receiver Operating Characteristic (ROC) curve [16]. ROC curve is used to obtain a balance between the number of classes predicted as being fault prone, and the number of classes predicted as not being fault prone.

The paper is organized as follows: Section 2 reviews the key points of available literature in the domain. Section 3 explains the independent and dependent variables used in our study. The description of the metrics is also provided. Section 4 discusses the research methodology and gives the details of the data used for analysis. It also explains the various methods used and the performance evaluation measures. Section 5 analyzes the univariate and the multivariate results. We have compared our results with the results of the previous results in this section. The model predicted is evaluated using the ROC curve in Section 6. Finally, the work is concluded in Section 7.

2. LITERATURE REVIEW

Significant work has been done in the field of fault detection. The complete survey of fault prediction studies till 2008 is provided in the paper by C. Catal [17]. Highlights of select papers have been discussed in this section, including papers published post 2008. There are various categories of methods to predict faulty classes such as machine learning methods, statistical methods, etc. We have observed that much of the previous work used traditional statistical methods [18, 20, 21, 16] to bring out the results, but very few studies have used machine learning methods. Recently, the trend is shifting from traditional statistical methods to modern machine learning methods. The most common statistical methods used are univariate and multivariate logistic regression. A few key points of the papers using statistical methods are discussed. The paper by N. Ohlsson et al.[18] has worked on improving the techniques used by Khosgoftaar [19] (i.e., Principal Component Analysis and Discriminant Analysis). This paper [18] has discussed some problems that were faced while using these methods and thus suggested remedies to those problems. Another approach to identify faulty classes early in the development cycle is to construct prediction models. The paper [20] has constructed a model to predict faulty classes using the metrics that can be collected during the design stage. This model has used only object oriented design metrics. Tang et al. [21] conducted an empirical study on three industrial real time systems and validated the CK [5] object oriented metric suite. They found that only WMC and RFC are strong predictors of faulty classes. They have also proposed a new set of metrics, which are useful indicators of object oriented fault prone classes. It has been seen that most of the empirical studies have ignored the confounding effect of class size while validating the metrics. Various studies [6, 11, 22] have shown that class size is associated with many contemporary object oriented metrics. Thus, it becomes important to revalidate contemporary object oriented metrics after controlling or taking into account the effect of class size [16]. The two papers by El. Emam et al. [16, 23] showed a strong size confounding effect and thus concluded that the metrics that were strongly associated with fault proneness before being controlled for size were not associated with fault proneness anymore after being controlled for size. Another empirical investigation [11] by M.Cartwright et al. conducted on a real time C++ system discussed the use of object oriented constructs such as inheritance and therefore polymorphism. M. Cartwright et al. [11] have found high defect densities in classes that participated in inheritance as compared to classes that did not. The probable reasons for this observation have been discussed in the paper. Briand et al. [1] have empirically investigated 49 metrics (28 coupling measures, 10 cohesion measures, and 11 inheritance measures) for predicting faulty classes. There were 8 systems being studied (consisting of 180 classes in all), each of which was a medium sized management information system. They used univariate and multivariate analysis to find the individual and the combined effect of object oriented metrics and fault proneness. They did not examine the LCOM metric and found that all the other metrics are strong predictors of fault proneness except for NOC. Another paper by Briand et al. [24] has also validated the same 49 metrics. The system used for this study was the multi-agent development system, which consists of three classes. They found NOC metric to be insignificant, while DIT was found to be significant in an inverse manner. WMC, RFC, and CBO were found to be strongly significant. Yu et al. [25] empirically tested 8 metrics in a case study in which the client side of a large network service management system was studied. The system is written in Java and consists of 123 classes. The validation was carried out using regression analysis and discriminant analysis. They found that all the metrics were significant predictors of fault proneness except DIT, which was found to be insignificant.

Recently, researchers have also started using some machine learning techniques to predict the model. Gyimothy et al. [12] calculated CK [5] metrics from an open source web and email suite called Mozilla. To validate the metrics, regression and machine learning methods (decision tree and artificial neural networks) were used. The results concluded NOC to be insignificant, whereas all the other metrics were found to be strongly significant. Zhou et al. [26] have used logistic regression and machine learning methods to show how object oriented metrics and fault proneness are related when fault severity is taken into account. The results were calculated using the CK metrics suite and were based on the public domain NASA dataset. WMC, CBO, and SLOC were found to be strong predictors across all severity levels. Prior to this study, no previous work had assessed severity of faults. The paper by S. Kanmani et al. [13] has introduced two neural network based prediction models. The results were compared with two statistical methods and it was concluded that neural networks performed better as compared to statistical methods. Fenton et al. [27] introduced the use of bayesian belief networks (BBN) for the prediction of faulty classes. G.J. Pai et al. [2] also built a bayesian belief network (BN) and showed that the results gave comparable performance with the existing techniques. I. Gondra [14] has performed a comparison between the artificial neural network (ANN) and the support vector machine (SVM) by applying them to the problem of classifying classes as faulty or non-faulty. Another goal of this paper was to use the sensitivity analysis to select the metrics that are more likely to indicate the errors. After the work of Zhou et al. [26], the severity of faults was taken into account by Shatnawi et al. [28] and Singh et al. [4]. Shatnawi et al. used the subset of CK [5] and Lorenz & Kidd [9] metrics to validate the results. The data was collected from three releases of the Eclipse project. They concluded that the accuracy of prediction decreases from release to release and some alternative methods are needed to get more accurate prediction. The metrics, which were found to be very good predictors across all versions and across all severity levels, were WMC, RFC, and CBO. Singh et al. [4] used the public domain NASA dataset to determine the effect of metrics on fault proneness at different severity levels of faults. Machine learning methods (decision tree and artificial neural network networks) and statistical method (logistic regression) were used. The predicted model showed lower accuracy at a high severity level as compared to medium and low severities. It was also observed that performance of machine learning methods was better than statistical methods. Amongst all the CK metrics used CBO, WMC, RFC, and SLOC showed the best results across all the severity levels of faults. Malhotra et al. [29] have used LR and 7 machine learning techniques (i.e., artificial neural networks, random forest, bagging, boosting techniques [AB, LB], naive bayes, and kstar) to validate the metrics. The predicted model using LB technique showed the best result and the model predicted using LR showed low accuracy.

From the survey we have conducted, the following observations were made:

- We observed that among the number of metrics available in literature, the CK metric suite is most widely used. It has been seen that most of the studies have also defined their own metric suite and they have used them for carrying out the analysis.
- Among the various categories of methods available to predict the most accurate model such as machine learning methods, statistical methods, etc. the trend is shifting from the traditional statistical methods to the machine learning methods. It has been observed that machine learning is widely used in new bodies of research to predict fault prone classes. Results of various studies also show that better results are obtained with machine learning as compared to statistical methods.

- Papers have used different types of datasets, which are mostly public datasets, commercial datasets, open source, or students/university datasets. We have observed that the public datasets, which have been mostly used in the studies, are from the PROMISE and NASA repositories.

3. Dependent and Independent Variables

In this section, we present the independent and dependent variables used in this study along with a summary of the metrics studied in this paper.

In this paper, we have used object-oriented metrics as independent variables. A summary of the metrics used in this paper is given in Table 1. The dependent variable is fault proneness. Fault proneness is defined as the probability of fault detection in a class [1, 2, 3, 4]. We have

Table 1. Metrics Studied

S.No.	Metric	Definition
1.	WMC - Weighted methods per class	The WMC metric is the sum of the complexities of all methods in a class. Complexity can be measured in terms of cyclomatic complexity, or we can arbitrarily assign a complexity value of 1 to each method. The Ckjm program assigns a complexity value of 1 to each method. Therefore, the value of the WMC is equal to the number of methods in the class.
2.	DIT - Depth of Inheritance Tree	The Depth of Inheritance Tree (DIT) metric for each class is the maximum number of steps from the class node to the root of the tree. In Java, where all the classes inherit the object, the minimum value of the DIT is 1.
3.	NOC - Number of Children	A class' Number of Children (NOC) metric measures the number of immediate descendants of the class.
4.	CBO - Coupling Between Object classes	The CBO for a class represents the number of classes to which it is coupled and vice versa. This coupling can occur through method calls, field accesses, inheritance, arguments, return types, and exceptions.
5.	RFC - Response for a Class	The value of RFC is the sum of the number of methods called within the class' method bodies and the number of the class' methods.
6.	LCOM - Lack of Cohesion in Methods	LCOM measures the dissimilarity of methods in a class by looking at the instance variables used by the methods in that class.
7.	Ca - Afferent couplings (not a C&K metric)	A class' afferent couplings are the number of other classes that use a specific class.
8.	Ce - Efferent couplings (not a C&K metric)	A class' efferent couplings are the number of other classes that are used by the specific class.
9.	NPM - Number of Public Methods (not a C&K metric; CIS: Class Interface Size in the QMOOD metric suite)	The NPM metric counts all the methods in a class that are declared as being public.
10.	LCOM3 -Lack of cohesion in methods Henderson-Sellers version	LCOM3 varies between 0 and 2. m - number of procedures (methods) in class a - number of variables (attributes in class μ(A) - number of methods that access a variable (attribute) $\text{LCOM3} = \frac{\left(\frac{1}{a}\sum_{j=1}^{a}\mu(A_j)\right) - m}{1 - m}$ The constructors and static initializations are taken into account as separate methods.

Table 1. Metrics Studied

S.No.	Metric	Definition
11.	LOC - Lines of Code (not a C&K metric)	The lines of code is calculated as the sum of the number of fields, the number of methods, and the number of instructions in a given class.
12.	DAM: Data Access Metric (QMOOD metric suite)	This metric is the ratio of the number of private (protected) attributes to the total number of attributes declared in the class. A high value is desired for DAM. (Range 0 to 1)
13.	MOA: Measure of Aggregation (QMOOD metric suite)	The count of the number of data declarations (class fields) whose types are user defined classes.
14.	MFA: Measure of Functional Abstraction (QMOOD metric suite)	This metric is the ratio of the number of methods inherited by a class to the total number of methods accessible by member methods of the class. The constructors and the java.lang.Object (as parent) are ignored. (Range 0 to 1)
15.	CAM: Cohesion Among Methods of Class (QMOOD metric suite)	The metric is computed using the summation of the number of different types of method parameters in every method divided by a multiplication of a number of different method parameter types in whole class and the number of methods. A metric value close to 1.0 is preferred. (Range 0 to 1).
16.	IC: Inheritance Coupling (quality oriented extension for the C&K metric suite)	This metric provides the number of parent classes to which a given class is coupled. A class is coupled to its parent class if one of the following conditions is satisfied: • One of its inherited methods uses a variable (or data member) that is defined in a new/redefined method. • One of its inherited methods calls a method that is defined in the parent class. • One of its inherited methods is called by a method that is defined in the parent class and uses a parameter that is defined in that method.
17.	CBM: Coupling Between Methods (quality oriented extension for the C&K metric suite)	The metric measures the total number of new/redefined methods to which all the inherited methods are coupled.
18.	AMC: Average Method Complexity (quality oriented extension to C&K metric suite)	This metric measures the average method size for each class. The size of a method is equal to the number of Java binary codes in the method.
19.	CC - McCabe's Cyclomatic Complexity	It is equal to the number of different paths in a method (function) plus one. The cyclomatic complexity is defined as: CC = E - N + P where: E - the number of edges of the graph N - the number of nodes of the graph P - the number of connected components

used logistic regression and machine learning methods, which are based on predicting probabilities [1-4]

The program Ckjm calculates six object oriented metrics specified by Chidamber and Kemerer by processing the bytecode of compiled Java files. It also calculates a few of the other metrics. Ckjm follows the UNIX tradition of doing one thing well. [30]

4. Research Methodology

In this section we present the descriptive statistics for all the metrics that we have considered. We have also explained the methodology used (i.e., one statistical method and six machine learning methods). The performance evaluation measures are also presented.

4.1 Empirical Data Collection

This study makes use of an Open Source dataset "Apache POI" [15]. Apache POI is a pure Java library for manipulating Microsoft documents. It is used to create and maintain Java API for manipulating file formats based upon the office open XML standards (OOXML) and Microsoft OLE2 compound document format (OLE2). In short, we can read and write MS Excel files using Java. In addition, we can also read and write MS word and MS PowerPoint files using Java. The important use of the Apache POI is for text extraction applications such as web spiders, index builders, and content management systems. This system consists of 422 classes. Out of 422 classes, there are 281 faulty classes containing 500 numbers of faults. It can be seen from Fig. 1 that 71.53% of classes contain 1 fault, 15.3 % of classes contain 2 faults and so on. As shown in the pie chart, the majority of classes consist of 1 fault. Table 2 summarizes the distribution of faults and faulty classes in the dataset.

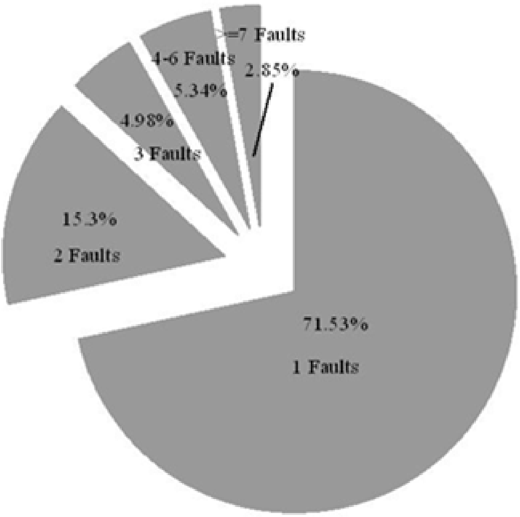


Fig. 1. Distribution of Faults

Table 2. Data Description

No. of faulty classes	281
% of faulty classes	63.57
No. of faults	500
Language used	Java

4.2 Descriptive Statistics

Table 3 shows the “mean," “median," “min," “max," “std dev," “25% quartile," “50% quartile,” and “75% quartile” of all the independent variables used in our study. We can make the following observations from Table 3.

Table 3. Descriptive Statistics

Metric	Mean	Std. Error of Mean	Median	Std. Deviation	Minimum	Maximum	Percentiles		
							25	50	75
WMC	13.501	0.698	10	14.677	0	134	5	10	16
DIT	1.869	0.040	2	0.850	1	6	1	2	2
NOC	0.738	0.331	0	6.963	0	134	0	0	0
CBO	10.120	0.932	6	19.585	0	214	4.75	6	9
RFC	30.351	1.763	21	37.067	0	390	13	21	36.25
LCOM	100.464	21.017	22	441.849	0	7059	1	22	53.25
CA	5.233	0.838	2	17.620	0	212	1	2	4
CE	5.224	0.431	4	9.059	0	133	2	4	6
NPM	11.600	0.606	9	12.747	0	101	4	9	14
LCOM3	0.999	0.025	0.85	0.534	0	2	0.749	0.85	1.129
LOC	292.595	30.046	124.5	631.675	0	9886	59.75	124.5	321.25
DAM	0.459	0.019	0.5	0.404	0	1	0	0.5	0.889
MOA	0.814	0.121	0	2.551	0	34	0	0	1
MFA	0.358	0.015	0.361	0.318	0	1	0	0.361	0.572
CAM	0.376	0.010	0.311	0.208	0	1	0.253	0.311	0.467
IC	0.577	0.026	1	0.555	0	3	0	1	1
CBM	1.952	0.116	1	2.439	0	20	0	1	4
AMC	19.362	1.880	12.192	39.516	0	616.375	6.375	12.192	20.544
MAX_CC	3.704	0.367	2	7.713	0	126	1	2	3
AVG_CC	1.188	0.052	0.976	1.090	0	17.125	0.814	0.975	1.289

The size of a class measured in terms of lines of source code ranges from 0 to 9886. We can observe that the NOC metric values are 0 in 75% of the classes. Also, the DIT metric values are low, the biggest DIT metric value is 6, and 75% of the classes have 2 levels of inheritance at most. This shows that inheritance is not used much in the system. Similar results were also observed by other authors [1, 11].There is a high cohesion observed in the system. The cohesion metrics (i.e., LCOM and LCOM3) have high values. The value of LCOM metric ranges from 0 to 7,059 and the LCOM3 metric ranges from 0 to 2 (which is the maximum LCOM3 value).

4.3 Methods Used

In this study, we have used one statistical model and six machine learning models to predict a fault proneness model.

4.3.1 The statistical model

Logistic regression is the commonly used statistical modelling method. Logistic regression is used to predict the dependent variable from a set of independent variables (a detailed description is given by [3, 31, 32]). It is used when the outcome variable is binary or dichotomous. We have used both univariate and multivariate regression. Univariate logistic regression finds the relationship between the dependent variable and each independent variable. It finds whether there is any significant association between them. Multivariate logistic regression is done to construct a prediction model for the fault proneness of classes. It analyzes which metrics are useful when they are used in combination. Logistic regression results in a subset of metrics that have significant parameters. To find the optimal set of independent variables (metrics), there are two stepwise selection methods, which are forward selection and backward elimination [32]. Forward selection examines the variables that are selected one at a time for entry at each step. The backward elimination method includes all the independent variables in the model and the variables are deleted one at a time from the model until the stopping criteria is fulfilled. We have used the forward stepwise selection method.

The general multivariate logistic regression formula is as follows [3]:

$$\text{Prob}\ (X_1\ , X_2, \ldots, X_{n)} = \frac{e^{g(x)}}{1+e^{g(x)}}$$

where $g(x) = B_0 + B_1{*}X_1 + B_2{*}\ X_2 + \ldots + B_n{*}\ X_n$
‘prob’ is the probability of a class being faulty
$X_{i,}$ $(1\leq i \leq n)$ are independent variables

The following statistics are reported for each metric from the above formula:

1. Odds Ratio: The odds ratio is calculated using Bi's. The formula for the odds ratio is R= exp (Bi). This is calculated for each independent variable. The odds ratio is the probability of the event divided by the probability of a non-event. The event in our study is the probability of having a fault and the non- event is the probability of not having a fault [4].
2. Maximum Likelihood Estimation (MLE) and coefficients (Bi's): MLE is the likelihood function that measures the probability of observing a set of dependent variables [4]. MLE finds the coefficient in such a way that the log of the likelihood function is as large as possible. The more the value of the coefficient the more the impact of the independent variables on predicted fault proneness is.

4.3.2 Machine Learning Models

Besides the statistical approach, we have used six machine learning methods. All the methods can be used to predict fault proneness by using just one metric or by using a combination of metrics together for prediction [12]. We have used machine learning techniques to predict the accuracy of the models when a combination of metrics is used. Not much of the work in the area of fault prediction is done using machine learning techniques. There are various machine learning techniques available. From amongst all of the methods, artificial neural networks (ANN) [33] and decision trees (DT) [34] have been widely used in literature [12, 13, 14, 4].The use of decision trees in predicting fault proneness has been proposed in Porter & Selly [35]. The paper [14] has used ANN to predict the value of a continuous measure of fault proneness. For performing the classification of classes as fault prone and non-fault prone, the paper [14] has used a support vector machine (SVM). The application of SVMs to the fault proneness prediction problem has been explained by Xing et al. [36]. The paper [29] has used ANN, random forest, bagging, boosting, and some more machine learning techniques in order to predict the faulty classes.

There are various variants of boosting algorithms available, but the authors have used two variants (i.e., AB [37] and LB [38]), which have been designed for classification purposes. In literature, boosting algorithms were not evaluated, but this paper [29] shows that the boosting technique LB gave the best results in terms of AUC. Thus, the authors concluded that boosting techniques may be effective in predicting faulty classes.

To predict the fault proneness of classes, we have used the following machine learning methods, and these machine learning algorithms are available in the WEKA open source tool [39]:

a. *Random Forest:* A random forest is made up of a number of decision trees. Each decision tree is made from a randomly selected subset of the training dataset using replacement. For building a decision tree, a random subset of available variables is used. This helps us to choose how best to partition the dataset at each node. The final result/outcome is chosen by the majority. Each decision tree in the random forest gives out its own vote for the result and the majority wins. In building a random forest, we can mention the number of decision trees we want in the forest. Each decision tree is built to its maximum size. There are various advantages of a random forest. Very little pre-processing of data is required. Also, we do not need to do any variable selection before starting to build the model. A random forest itself takes the most useful variables [40].

b. *Adaboost*: Adaboost is short for adaptive boosting. It is a machine learning algorithm that can be used along with many other learning algorithms. This leads to an improvement in efficiency and performance. Adaboost is adaptive as it adapts to the error rates of the individual weak hypothesis. Also, adaboost is a boosting algorithm as it can efficiently convert a weak learning algorithm into a strong learning algorithm. Adaboost calls a given weak algorithm repeatedly in a series of rounds. The important concept for an adaboost algorithm is to maintain a distribution of weights over the training set. Initially all the weights are equal but on each round the weights of incorrect classified examples are increased so that a weak learner is forced to focus on the hard examples in the training set. This is how a weak learning algorithm is changed to a strong learning algorithm. Adaboost is less susceptible to an over fitting problem than most learning algorithms [40].

c. *Bagging*: Bagging, which is also known as bootstrap aggregating, is a technique that repeatedly samples (with replacement) from a data set according to a uniform probability distribution [41]. Each bootstrap sample has the same size as the original data. Because the sampling is done with replacement, some instances may appear several times in the same training set, while others may be omitted from the training set. On average, a bootstrap sample D_i contains approximately 63% of the original training data because each sample has a probability $1- (1- 1/N)^N$ of being selected in each D_i. If N is sufficiently large, this probability converges to $1-1/e = 0.632$. After training the k classifiers, a test instance is assigned to the class that receives the highest number of votes [42].

d. *Multilayer Perceptron*: Multilayer Perceptron (MLP) is an example of an artificial neural network. It is used for solving different problems, example pattern recognition, interpolation, etc. It is an advancement to the perceptron neural network model. With one or two hidden layers, they can solve almost any problem. They are feedforward neural networks trained with the back propagation algorithm. Error back-propagation learning consists of two passes: a forward pass and a backward pass. In the forward pass, an input is presented to the neural network, and its effect is propagated through the network layer by layer. Dur-

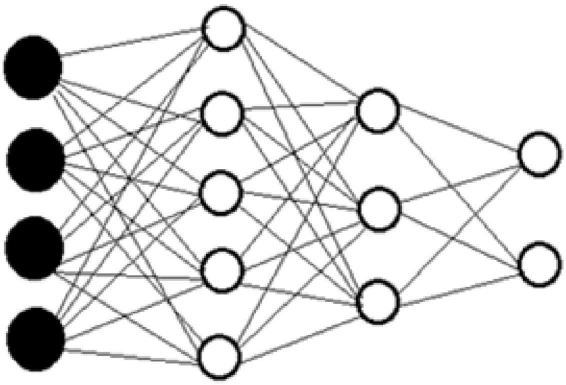


Fig. 2. Multilayer Perceptron

ing the forward pass the weights of the network are all fixed. During the backward pass the weights are all updated and adjusted according to the error computed. An error is composed from the difference between the desired response and the system output. This error information is fed back to the system and adjusts the system parameters in a systematic fashion (the learning rule). The process is repeated until the performance is acceptable [42].

e. *Support Vector Machine*: A Support Vector Machine (SVM) is a learning technique that is used for classifying unseen data correctly. For doing this, SVM builds a hyperplane, which separates the data into different categories. The dataset may or may not be linearly separable. By "linearly separable" we mean that the cases can be completely separated (i.e., the cases with one category are on the one side of the hyperplane and the cases with the other category are on the other side). For example, Fig. 3 shows the dataset where examples belong to two different categories - triangles and squares. Since these points are represented on a 2-dimensional plane, a 1-dimensional line can separate them. To separate these points into 2 different categories, there are an infinite number of lines possible. Two possible candidate lines are shown in Fig. 3. However, only one of the lines gives a maximum separation/margin and that line is selected. "Margin" is defined as the distance between the dashed lines (as shown in Fig. 3), which is drawn parallel to the separating lines. These dashed lines give the distance between the separating line and closest vectors to the line. These vectors are called support vectors. SVM can also be extended to the non-linear

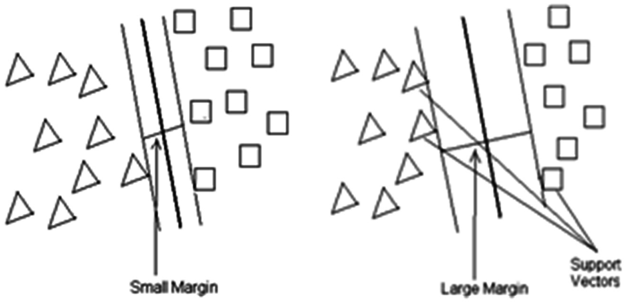


Fig. 3. Support Vector Machine

boundaries by using the kernel trick. The kernel function transforms the data into a higher dimensional space to make the separation easy. [16]

f. *Genetic Programming*: Genetic Programming is a branch of genetic algorithms. It is inspired by biological evolution. Genetic Programming creates computer programs that can perform a user defined task. For doing this, the following 4 steps are used:

i. First, all the computer programs are made.
ii. Then, each program is executed and assigned a fitness value according to how well it solves the problem.
iii. Then, a new population of computer programs is created:
   • From among all the programs the best existing programs are copied.
   • Mutation is carried out to create new programs.
   • Crossover is also carried out to create new programs.
iv. Finally, the best computer program created so far in any generation is the result of Genetic Programming.

4.4 Performance Evaluation Measures

To measure the performance of the predicted model, we have used the following performance evaluation measures:

Sensitivity: It measures the correctness of the predicted model. It is defined as the percentage of classes correctly predicted to be fault prone. Mathematically,
Sensitivity = ((Number of modules correctly predicted as fault prone) / (total number of actual faulty modules)) * 100

Specificity: It also measures the correctness of the predicted model. It is defined as the percentage of classes predicted that will not be fault prone. Mathematically,
Specificity = ((Number of modules correctly predicted as non- fault prone) / (total number of actual non faulty modules)) * 100

Precision or Accuracy: It is defined as the ratio of number of classes (including faulty and non- faulty) that are predicted correctly to the total number of classes.

Receiver Operating Characteristic (ROC) analysis: The performance of the outputs of the predicted models was evaluated using ROC analysis. It is an effective method of evaluating the performance of the model predicted. The ROC curve is defined as a plot of sensitivity on the y-coordinate versus its 1-specificity on the x-coordinate [16]. While constructing ROC curves, we selected many cutoff points between 0 and 1, and calculated sensitivity and specificity at each cutoff point. The ROC curve is used to obtain the required optimal cutoff point that maximizes both sensitivity and specificity [16, 4].

The *validation method* used in our study is k-cross validation (the value of k is taken as 10) in which the dataset is divided into approximately equal k partitions [43]. One partition at a time is used for testing the model and the remaining k-1 partitions are used for training the model. This is repeated for all the k partitions.

5. RESULT ANALYSIS

In this section, we have analyzed the results of our study. In this study, we have validated the CK metric suite. To begin with the data analysis, the first step is to identify the subset of the object oriented metrics that are related to fault proneness and that are orthogonal to each other. The statistical modeling technique used for this purpose is univariate logistic regression. After identifying a subset of metrics, we have used the multivariate logistic regression technique to construct a multivariate model that can be used to predict the overall fault in the system. To predict the best model that gives the highest accuracy we have used various machine learning techniques. We performed the analysis of an Open Source software, poi [15], which consisted of 422 classes (see Section 4.1). The performance of each of the predicted models was determined using several performance measures (i.e., sensitivity, specificity, precision, and the ROC analysis).

5.1 Univariate LR Analysis Results

We conducted univariate analysis to find whether each of the metrics (independent variables) is significantly associated with fault proneness (dependent variable). Table 4 represents the results of univariate analysis. It provides the coefficient (B), standard error (SE), statistical significance (sig.), and odds ratio (exp (B)) for each metric [4]. The parameter "sig" tells whether each of the metric is a significant predictor of fault proneness. If the "sig" value of a metric is below or at the significance threshold of 0.01, then the metric is said to be significant in predicting the

Table 4. Univariate Analysis

S.no	Metric	B	SE	Sig.	Exp(B)
1	WMC	0.123	0.018	**0.000**	1.131
2	DIT	-0.188	0.115	0.102	0.828
3	NOC	0.003	0.015	0.835	1.003
4	CBO	0.056	0.020	**0.004**	1.057
5	RFC	0.055	0.008	**0.000**	1.056
6	LCOM	0.012	0.003	**0.000**	1.012
7	CA	0.007	0.007	0.354	1.007
8	CE	0.251	0.043	**0.000**	1.285
9	NPM	0.109	0.018	**0.000**	1.115
10	LCOM3	-0.943	0.192	**0.000**	0.389
11	LOC	0.004	0.001	**0.000**	1.004
12	DAM	1.477	0.264	**0.000**	4.381
13	MOA	0.495	0.128	**0.000**	1.641
14	MFA	-0.004	0.311	0.991	0.996
15	CAM	-3.844	0.568	**0.000**	0.021
16	IC	1.460	0.206	**0.000**	4.307
17	CBM	0.511	0.065	**0.000**	1.668
18	AMC	0.013	0.006	**0.036**	1.013
19	MAX_CC	0.187	0.045	**0.000**	1.206
20	AVG_CC	0.828	0.192	**0.000**	2.289

faulty classes [4]. Table 4 shows the significant values in bold. The coefficient "(B)" shows the strength of the independent variable. The higher the value, the higher the impact of the independent variable is. The sign of the coefficient tells whether the impact is positive or negative. We can see that DIT, NOC, Ca, and MFA metrics are not significant and are therefore not taken for any further analysis. Thus, in this way we can reduce the number of independent variables and select only the best fault predictors. The following notations used in tables 5-9 shows the degree of the significance:

++ shows the significance of the metric at 0.01, + shows the significance of the metric at 0.05, -- shows the significance of the metric at 0.01 but in an inverse manner, − shows the significance of the metric at 0.05 but in an inverse manner, and 0 shows that the metric is insignificant.

Table 5. Univariate Results of Size Metrics

Metric	Notation
WMC	++
NPM	++
LOC	++
DAM	++
MOA	++
AMC	+

Table 6. Univariate Results of Coupling Metrics

Metric	Notation
RFC	++
CBO	+
CA	0
CE	++
IC	++
CBM	++

Table 7. Univariate Results of Cohesion Metrics

Metric	Notation
LCOM	++
LCOM3	--
CAM	--

Table 8. Univariate Results of Inheritance Metrics

Metric	Notation
DIT	0
NOC	0
MFA	0

Table 9. Univariate Results of the Complexity Metric

Metric	Notation
CC	++

5.2 Multivariate LR Analysis Results

Multivariate analysis is done to find the combined effect of all of the metrics together on fault proneness. For doing multivariate analysis, we have used forward stepwise selection to determine which variables should be included in the multivariate model. Out of all the variables, one variable in turn is selected as the dependent variable and the remaining others are used as independent variables [44]. In univariate analysis 16 metrics were found to be significant. Table 10 shows the results of the multivariate model. The coeff (B), statistical significance (Sig.), standard error (SE), and odds ratio (Exp (B)) are also shown in the table for all the metrics included in the model. We can see that only 3 metrics (i.e., DIT, RFC, and CBM) are included in the model.

Table 10. Multivariate Model Statistics

Metric	B	SE	Sig.	Exp(B)
DIT	-0.522	0.165	0.002	0.594
RFC	0.031	0.007	0.000	1.032
CBM	0.531	0.078	0.000	1.701
CONSTANT	-0.089	0.328	0.785	0.914

5.3 Obtaining a Relationship Between Object Oriented Metrics and Fault Proneness

In this section, we have discussed our results and also we have compared our results with the results of previous studies shown in Table 11.

Table 11. Results of Different Validation

Metric	Our results	Basili et al. (1996)	Tang et al. (1999)	Briand et al. (2000)	Briand et al. (2001)	El Emam et al. (2001)		Yu et al. (2002)	Gyimothy et al. (2005)	Zhou et al. (2006)		Olague et al. (2007)		
Lang. used	Java	C++	C++	C++	C++	C++		Java	C++	C++		Java		
Method used	LR,ML (RF,Ab,MLP, Bagging, SVM,GP)	LR	LR	LR	LR	LR		OLS	LR,ML (DT,ANN)	LR,ML (NNage, RF,NB)		LR		
Type of data		Univ.	Comm.	Univ.	Comm.	Comm.		Comm.	Open source	NASA dataset		Open source		
Fault severity taken	No	No	No	No	No	No		No	No	Yes		No		
						#1	#2			LSF/USF	HSF	R3	R4	R5
WMC	++	+	+	+	++	+	0	++	++	++	++	++	++	++
DIT	0	++	0	++	--	0	0	0	+	0	0	0	--	0
RFC	++	++	+	++	++	++	0	+	++	++	++	++	++	++
NOC	0	--	0	-	0			++	0	--		0	0	0
CBO	++	+	0	++	++	+	0	+	++	++	++	++	++	0
LCOM	++	0							+	+	++	++	++	++
SLOC	++			++		++	++		++	++	++			

Table 11. Results of Different Validation (cont'd…)

Metric	Shatnawi et al. (2008)												Aggarwal et al. (2008)	English et al. (2009)	Singh et al. (2009)				Zhou et al. (2010)			Burrows et al. (2010)		
Lang. used	java												Java	Java	C++				Java			1.Java 2.Java, AspectJ 3.Java AspectJ		
Method used	LR												LR	LR	LR,ML (DT,ANN)				LR			LR		
Type of data	Open source												Univ.	Open source	NASA dataset				Open source			1.open source 2.web based 3. S/w prod.		
Fault severity taken	No(UBA)			Yes(UMA)									No	No	Yes				No			No		
	2.0	2.1	3.0	2.0			2.1			3.0					HSF	MSF	LSF	USF	2.0	2.1	3.0	1.	2.	3.
				HSF	MSF	LSF	HSF	MSF	LSF	HSF	MSF	LSF												
WMC	++	++	++	++	++	++	++	++	++	++	++	++	++		++	++	++	++	++	++	++			
DIT	0	0	++	0	0	0	++	0	0	0	0	++	0	++	0	0	0	0				0	0	0
RFC	++	++	++	++	++	++	++	++	++	++	++	++	++	++	++	++	++	++						
NOC	0	0	0	0	0	0	0	0	0	0	0	0	0	++	0	--	0	--						
CBO	++	++	++	++	++	++	++	++	++	++	++	++	++	++	++	++	++	++						
LCOM													+		++	++	0	++						
SLOC													++	++	++	++	++	++	++	++	++			

++, Denotes the metric is significant at 0.01; +, denotes the metric is significant at 0.05; --, denotes the metric is significant at 0.01 but in an inverse manner; -, denotes the metric is significant at 0.05 but in an inverse manner; 0, denotes that the metric is not significant.

A blank entry means that our hypothesis was not examined or that the metric was calculated in a different way. LR, logistic regression; UMR, Univariate Multinomial Regression; UBR, Univariate Binary Regression; OLS, Ordinary Least Square; ML, Machine Learning; DT, Decision Tree; ANN, Artificial Neural Network; RF, Random Forest; NB, Naı¨ve Bayes ;MLP, Multilayer Perceptron; Ab, Adaboost; SVM, Support Vector Machine; GP, Genetic Programming; LSF, Low Severity Fault; USF, Ungraded Severity Fault; HSF, High Severity Fault; MSF, Medium Severity Faults; #1, without size control; #2, with size control; 2.0, Eclipse version 2.0; 2.1, Eclipse version 2.1; 3.0,Eclipse version 3.0; 1., iBATIS system; 2., HealthWatcher application; 3., MobileMedia system; R3,Rhino 15R3; R4, Rhino 15R4; R5, Rhino 15R5; comm.,commercial; univ., university

5.3.1 Discussion about our results

All the size metrics, except AMC, are significant at 0.01. AMC is significant at 0.05. Amongst the cohesion metrics, we can see that LCOM3 and CAM have negative coefficients indicating that they have a negative impact on fault proneness. By definition, if LCOM, LCOM3, and CAM are significant, it means that fault proneness increases with a decrease in cohesion. Since CAM and LCOM3 are negatively related to fault proneness, we can conclude that fault proneness decreases with the decrease in cohesion. We can observe that out of 3 cohesion metrics, the majority (i.e., 2) of the metrics are negatively related. All the coupling metrics, except CA, are found to be strongly relevant to determine the fault proneness of the class. CBO is not strongly related but it still has a positive impact. CA is not significant to fault proneness, meaning it has neither a positive nor a negative impact. None of the inheritance metrics is found to be significant. The complexity metrics CC is found to be strongly and positively related to fault proneness.

5.3.2 Discussion of previous studies

We have done the comparison of our results with the results of the previous studies. CBO was found to be a significant predictor in the majority of the studies except by Tang et al. (1999) [21], El Emam et al. (2001) [23], and Olague et al. (2007) [45]. In El Emam et al. [20], the results were analyzed for the projects with and without size control. When size control was not taken into account, then CBO was found to be insignificant. Similarly, Olague et al. [45] predicted the fault prone classes for various versions of RhiNo.For one of the versions, the CBO was found to be insignificant. RFC was also found to be a significant predictor of fault proneness in all the studies except by El Emam et al. (2001) [23] when size control was not considered. Most of the studies (i.e., Tang et al. (1999)[21], Briand et al. (2000) [1], Briand et al. (2001)[24], Yu et al.(2002)[25], Shatnawi et al. (2008)[28], English et al.(2009)[44], Zhou et al. (2010)[46], and Burrows et al. (2010)[47]) did not examine the LCOM metrics or they calculated it in a very different manner. Among the studies that examined LCOM, it was insignificant with Basili et al. (1996) [31] and Singh et al. (2009) [4] for the Low Severity Fault (LSF) prediction model. The metric NOC, which is not found to be a significant predictor in our study, showed a negative impact on fault proneness by Basili et al. (1996) [31], Briand et al. (2000) [1], and Zhou et al. (2006) [46] for the LSF prediction model and by Singh et al. (2009) [4] for the Medium Severity Fault (MSF) and Ungraded Severity Fault (USF) prediction model. For the remaining previous studies, NOC was not considered to be significant. NOC was found to be very significant in predicting faulty classes by Yu et al. (2002) [25] and English et al. (2009) [44]. SLOC is found to be strongly relevant to fault proneness in all the studies. Various studies (i.e. Tang et al. (1999) [21], El emam et al. (2001) [23], Yu et al. (2002) [25], Zhou et al. (2006) [46], Singh et al. (2009) [4], Burrows et al. (2010) [47], and Aggarwal et al. (2008) [3]) showed DIT results that were similar to our results. For Basili et al. (1996) [31], Briand et al. (2000) [1], Gyimothy et al. (2005) [12], and English et al. (2009) [44] was found to be positive significant predictor. WMC is also found to be quite significant in all the previous studies. Thus, we can conclude that WMC and SLOC have always been significant predictors. DIT is not much useful in predicting the faulty classes.

6. Model Evaluation using the ROC CURVE

This section presents and summarizes the result analysis. We have used various machine learning methods to predict the accuracy of fault proneness. The validation method which we have used is k cross-validation, with the value of k as 10.

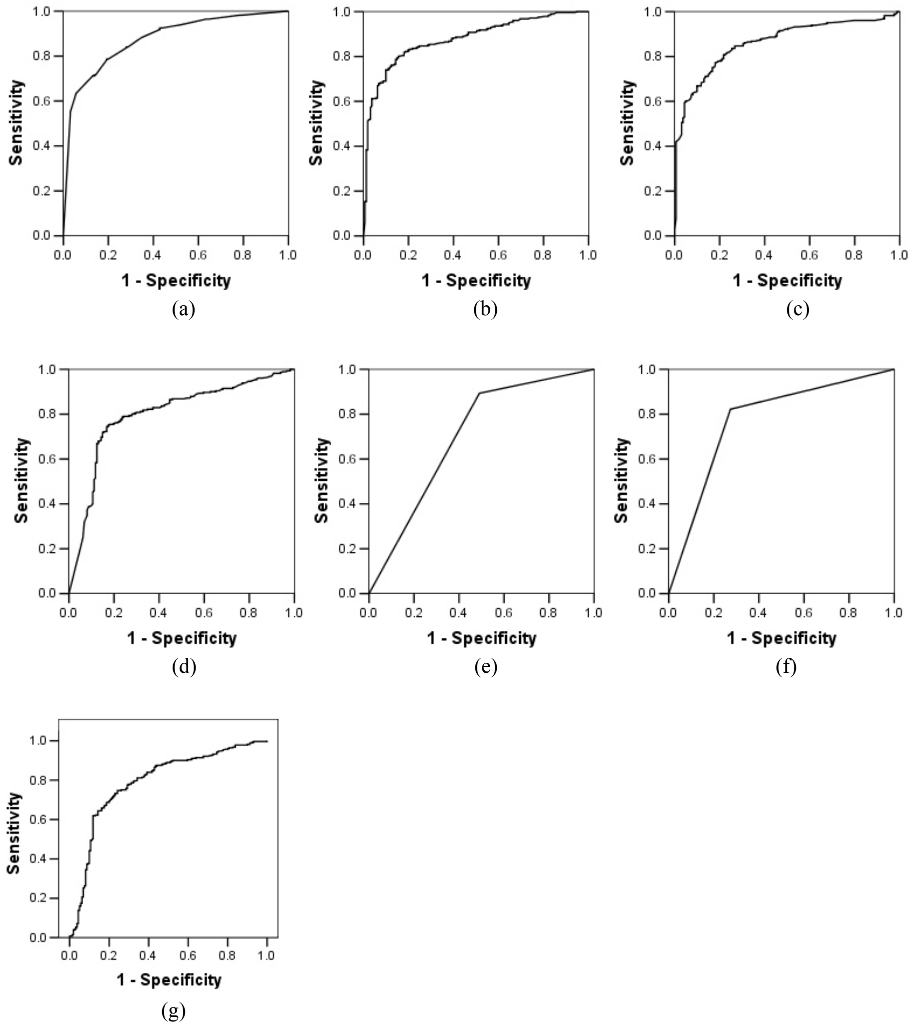
Table 12 summarizes the results of 10 cross-validation of the models predicted by using machine learning methods. It shows the sensitivity, specificity, precision, AUC, and the cutoff point for the model predicted using all the machine learning methods. We have used ROC analysis to find the cutoff point. The cutoff point is selected such that a balance is maintained between the number of classes predicted as being fault prone and not fault prone. The ROC

Table 12. Results of 10-cross Validation

S.No.	Method Used	Sensitivity	Specificity	Precision	Area under curve	Cut-off point
1	Random Forest	78.6	80.7	78.90	0.875	0.61
2.	Adaboost	80.8	78.3	79.86	0.861	0.62
3.	Bagging	82.9	80.1	81.99	0.876	0.57
4.	Multilayer Perceptron	77.6	77	77.25	0.799	0.54
5.	Support Vector Machine	89.3	51	76.30	0.70	0.5
6.	Genetic Programming	82.8	72.7	79.38	0.808	0.5
7.	Logistic Regression	74.7	73.9	74.4	0.791	0.59

curve is plotted with sensitivity on the y-axis and (1-specificity) on the y-axis. The point where sensitivity equals (1-specificity) is called the cutoff point. The ROC curves for the machine learning models are presented in Fig. 4.

We can see that the random forest and bagging give quite similar results. They show good results as compared to the results of the other methods. The specificity and AUC for both the models are quite similar. The specificity for the random forest is 80.7% whereas for bagging it is 80.1%. These values are quite high when compared to the values of the other methods. Also the ROC curve for the random forest and bagging gives high AUC values i.e. 0.875 and 0.876 respectively. The sensitivity of the random forest is 98.6%, whereas bagging shows a high sensitivity of 82.9%. The highest sensitivity is shown by the SVM method, which is 89.3%, but it



(a) (b) (c)

(d) (e) (f)

(g)

Fig. 4. ROC curve for (A) Adaboost, (B) Random Forest, (C) Bagging, (D) Multilayer Perceptron, (E) Genetic Programming, (F) SVM, (G) Logistic Regression

gives the lowest specificity of 51%. Also the AUC for the SVM model is 0.70. Thus, this method is not considered to be good. Adaboost and Genetic Programming show average results with a sensitivity of 80.8% and 82.8% respectively, with a specificity of 78.3% and 72.7%. Besides these machine learning models, we have also used a statistical method (i.e., logistic regression). We can observe that the sensitivity of logistic regression is the lowest as compared to other machine learning methods. Also, specificity is quite low when compared with most of the other machine learning methods. Thus, we can conclude from the discussion that the machine learning methods give better results as compared to the statistical methods. From amongst the machine learning methods under consideration, random forest and bagging are the best predicted models.

7. Conclusion

In any software project, there can be a number of faults. It is very essential to deal with these faults and to try to detect them as early as possible in the lifecycle of the project development. Thus, various techniques are available for this purpose in the literature, but previous research has shown that the object oriented metrics are useful in predicting the fault proneness of classes in object oriented software systems. The data is collected from an Open Source software Apache POI, which was developed in Java and consists of 422 classes. In this study, we have used object oriented metrics as the independent variables and fault proneness as the dependent variable. We have studied 19 object oriented metrics for predicting the faulty classes. Out of 19 metrics, we have identified a subset of metrics, which are significant predictors of fault proneness. For doing this, we have used univariate logistic regression. It was found that the metrics DIT, NOC, Ca, and MFA are not significant predictors of fault proneness and the remaining metrics that we have considered are found to be quite significant. We have also compared our results with those of previous studies and concluded that WMC and SLOC are significant predictors in the majority of the studies. After identifying a subset of metrics, we constructed a model that could predict the faulty classes in the system. Using multivariate analysis, we constructed the model in which only 3 metrics were included (i.e., DIT, RFC, and CBM). To predict the best model, we used six machine learning techniques that measured the accuracy in terms of sensitivity, specificity, precision, and AUC (Area Under the Curve). The cutoff point was also selected such that a balance is maintained between the number of classes predicted as fault and not fault prone. The ROC curve was used to calculate the cutoff point. We observed that the random forest and bagging gave the best results as compared to other models. Thus, we can conclude that practitioners and researchers may use bagging and the random forest for constructing the model to predict the faulty classes. The model can be used in the early phases of software development to measure the quality of the systems.

More similar type of studies can be carried out on different datasets to give generalized results across different organizations. We plan to replicate our study on larger datasets and industrial object oriented software systems. In future studies, we will take into account the severity of faults to get more accurate and efficient results. In this study, we have not taken into account the effect of size on fault proneness. In future work, we will also take into account some of the product properties such as size, and also process and resource related issues like the experience of people, the development environment, etc., which all effect fault proneness.

REFERENCES

[1] L. Briand, W. Daly and J. Wust, "*Exploring the relationships between design measures and software quality*," *Journal of Systems and Software*, Vol.51, No.3, 2000, pp.245-273.

[2] G. Pai, "*Empirical analysis of software fault content and fault proneness using Bayesian methods*," *IEEE Transactions on Software Eng.*, Vol.33,No.10,2007, pp.675-686.

[3] K. K. Aggarwal, Y. Singh, A. Kaur, and R. Malhotra, "*Empirical analysis for investigating the effect of object-oriented metrics on fault proneness: A replicated case study*," *Software Process: Improvement and Practice*, Vol.16,No.1,2009,pp.39-62.

[4] Y. Singh, A. Kaur, and R. Malhotra, "*Empirical vlidation of object-oriented metrics for predicting fault proneness models*," *Software Quality Journal*, Vol.18,No.1, 2010,pp.3-35.

[5] S. Chidamber and C. Kemerer, "*A Metrics Suite for Object-Oriented Design*," *IEEE Trans. Soft Ware Eng.*, Vol.20, No.6, 1994, pp.476-493.

[6] L.Briand, P. Devanbu, W. Melo, "*An investigation into coupling Measures for C++*," *In Proceedings of the 19th International Conference on Software Engineering.*

[7] J. Bansiya and C. Davis, "*A Hierarchical Model for Object-Oriented Design Quality Assessment*," *IEEE Trans. Software Eng.,* Vol.28, No.1, 2002, pp.4-17.

[8] F. Brito e Abreu and W. Melo, "*Evaluating the Impact of Object-Oriented Design on Software Quali ty*," *Proceedings Third Int'l Software Metrics Symposium*, 1996, pp.90-99.

[9] M.Lorenz and J. Kidd, "*Object-Oriented Software Metrics*," Prentice-Hall, 1994.

[10] W. Li and W. Henry, "*Object-Poiented Metrics that Predict Maintainability*," In Journal of Software and Sytems, 1993, Vol.23, pp.111-122.

[11] M.Cartwright and M. Shepperd, "*An empirical investigation of an object-oriented software system*," *IEEE Transactions on Software Engineering*, Vol.26, No.8,1999, pp.786-796.

[12] T.Gyimothy, R. Ferenc, and I.Siket, "*Empirical validation of object-oriented metrics on open source software for fault prediction*," *IEEE Transactions on Software Engineering*, Vol.31, No.10, 2005, pp.897-910.

[13] S. Kanmani, V.R. Uthariaraj, V. Sankaranarayanan, P. Thambidurai, "*Object-oriented software prediction using neural networks*," *Information and Software Technology*, Vol.49, 2007, pp.482-492.

[14] I. Gondra, "*Applying machine learning to software fault-proneness prediction*," *The Journal of Systems and Software*," Vol.81, 2008, pp.186-195.

[15] Promise. http://promisedata.org/repository/.

[16] K. El Emam, S. Benlarbi, N. Goel, and S. Rai, "A *validation of object-oriented metrics*," *NRC Technical report ERB-1063,*1999.

[17] C. Catal and B. Diri, "A *systematic review of software fault prediction studies*," *Expert Systems with Applications* Vol.36, 2009, pp 7346-7354.

[18] N. Ohlsson, M. Zhao and M. Helander, M, "*Application of multivariate analysis for soft ware fault prediction*," *Software Quality Journal*, Vol.7, 1998,pp.51-66.

[19] T.M. Khoshgoftaar, E.B. Allen, K.S. Kalaichelvan and N. Goel, "*Early quality prediction: a case study in telecommunications*," *IEEE Software*, Vol.13, No.1, 1996, pp.65-71.

[20] K.E. Emam and W. Melo, "*The Prediction of Faulty Classes Using Object-Oriented Design Metrics*," *Technical report: NRC 43609*, 1999.

[21] M.H. Tang, M.H. Kao, and M.H. Chen , "*An empirical study on object-oriented metrics*," *InProceedings of Metrics*, 242-249.

[22] L. Briand, J. Wuest, S. Ikonomovski, and H. Lounis, "A *comprehensive Investigation of Quality Factors in Object-Oriented Designs: An Industrial Case Study*," *International Software Engineering Research Network*, technical report ISERN-98-29, 1998.

[23] K. El Emam, S. Benlarbi, N. Goel, and S. Rai, "*The confounding effect of class size on the validity of object-oriented metrics*," IEEE Transactions on Software Engineering, Vol.27, No.7, 2001, pp.630-650.

[24] L. Briand, J. Wu¨st, J and H. Lounis, "*Replicated Case Studies for Investigating Quality Factors in Object-Oriented Designs*," *Empirical Software Engineering. International Journal (Toronto,Ont.),* Vol.6, No.1, 2001, pp.11-58.

[25] P. Yu, T. Systa, and H. Muller, "*Predicting fault-proneness using OO metrics: An industrial case study*," *In Proceedings of Sixth European Conference on Software Maintenance and Reengineering*,Budapest, Hungary, 2002, pp.99-107.

[26] Y. Zhou, and H. Leung, H, "*Empirical Analysis of Object-Oriented Design Metrics for Predicting High and Low Severity Faults*," *IEEE Transactions on Software Engineering*, Vol.32, No.10, 2006, pp.771-789.

[27] N. Fenton and N. Ohlsson, "*Quantitative analysis of faults and failures in a complex software system*," *IEEE Transactions on Software Engineering,* Vol.26, No.8, 2000, pp.797-814.

[28] R. Shatnawi and W. Li, "*The effectiveness of software metrics in identifying error-prone classes in post release software evolution process*," *The Journal of Systems and Software*,Vol.81, 2008,pp.1868-1882.

[29] R. Malhotra and Y. Singh, "*On the Applicability of Machine Learning Techniques for ObjectOriented Software Fault Prediction*," *Software Engineering: An International Journal*, Vol.1,No.1, 2011, pp.24-37.

[30] ckjm download : http://www.Spinellis.gr/sw/ckjm/

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

[32] D. Hosmer and S. Lemeshow, Applied logistic regression. New York: Wiley,1989.

[33] C.M. Bishop, "*Neural Networks for Pattern Recognition*," Oxford, U.K. : Claredon Press, 1995.

[34] J.R. Quinlan, C4.5 : Programs for Machine Learning. Morgan Kaufmann, 1993.

[35] A. Porter and R. Selly, "*Empirically guided Software Devlopment using Metric-Based Classification Trees,*" *IEEE Software*, Vol.7, No.2, 1990, pp.46-54.

[36] F. Xing, P. Gua, and M.R. Lyu, "*A novel method for early software quality prediction based on support vector machine*," *In: Proceedings of IEEE International Conference on Software Reliability Engineering*, 2005, pp.213-222.

[37] Y. Freund, R. Schapire, "*Experiments with a new boosting algorithm*," *In: Thirteenth International Conference on Machine Learning*, San Francisco, 1996, pp.148-156.

[38] J. Friedman, T. Hastie, and R. Tibshirani, "*Additive Logistic Regression: a Statistical View of Boosting*," Stanford University.

[39] Weka. Available: http://www.cs.waikato.ac.nz/ml/weka/

[40] Y. Freund and R.E. Schapire, "*A Short Introduction to Boosting*," *Journal of Japanese Society for Artificial Intelligence,* Vol.14, No.5, 1999, pp.771-780.

[41] L.Breiman, "*Bagging predictors*," *Machine Learning*, Vol.24, 1996, pp.123-140.

[42] R. Malhotra and A.Jain, "*Software Effort Prediction using Statistical and Machine Learning Me thod*," *International Journal of Advanced Computer Science and Applications* , Vol.2, No.1, 2011.

[43] M.Stone, "*Cross-validatory choice and assessment of statistical predictions*," *Journal Royal Stat. Soc*., Vol.36, 1974, pp.111-147.

[44] M.English, C.Exton, I.Rigon and B.Clearyp, "*Fault Detection and Prediction in an open source Software project*," Proceeding: *PROMISE '09 Proceedings of the 5th International conference on Predictor Models in Software Engineering*.

[45] H.Olague, L. Etzkorn, S. Gholston, and S.Quattlebaum, "*Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes*," *IEEE Transactions on Software Engineering*, Vol.33, No.8,2007, pp.402-419.

[46] Y.Zhou, B. Xu and H. Leung, "*On the ability of complexity metrics to predict fault-prone classes in object -oriented systems*," *The journal of Systems and Software*, Vol.83, 2010,pp.660-674.

[47] R. Burrows, F.C. Ferrari, O.A.L. Lemos, A. Garcia and F. Taiani, *"The impact of Coupling on the fault-Proneness of Aspect-oriented Programs:An Empirical Study," IEEE 21st Internati onal Symposium on Software Reliability Engineering*, 2010.

Ruchika Malhotra

She is an Assistant Professor in the Department of Software Engineering at Delhi Technological University (formerly known as Delhi College of Engineering) in Delhi, India. She is the Executive Editor of Software Engineering: An International Journal. She was an Assistant Professor at the University School of Information Technology of Guru Gobind Singh Indraprastha University in Delhi, India. Prior to joining the school, she worked as a full-time research scholar and received a doctoral research fellowship from the University School of Information Technology of Guru Gobind Singh Indraprastha in Delhi, India. She received her master's and doctorate degree in software engineering from the University School of Information Technology of Guru Gobind Singh Indraprastha University in Delhi, India. She is the co-author of the book titled Object Oriented Software Engineering, which was published by PHI Learning. Her research interests are in software testing, improving software quality, statistical and adaptive prediction models, software metrics, neural nets modeling, and the definition and validation of software metrics. She has published more for than 55 research papers in international journals and conferences. Malhotra can be contacted by e-mail at: ruchikamalhotra2004@yahoo.com

Ankita Jain

She is a research scholar with Delhi Technological University (formerly Delhi College of Engineering) in Delhi, India. She received her master's degree in Computer Technology and Applications (CTA) from Delhi Technological University. Her research interests are software quality, software metrics, and statistical and machine learning models. She has published papers in international journals/conferences. She can be contacted by e-mail at: ankita.bansal06@gmail.com.