Prediction of Software Quality Model Using Gene Expression Programming

Yogesh Singh, Arvinder Kaur, and Ruchika Malhotra

University School of Information Technology, GGS Indraprastha University,
Delhi 110403, India
Ys66@rediffmail.com, arvinderkaurtakkar@yahoo.com,
ruchikamalhotra@yahoo.com

Abstract. There has been number of measurement techniques proposed in the literature. These metrics can be used in assessing quality of software products, thereby controlling costs and schedules. The empirical validation of object-oriented (OO) metrics is essential to ensure their practical relevance in industrial settings. In this paper, we empirically validate OO metrics given by Chidamber and Kemerer for their ability to predict software quality in terms of fault proneness. In order to analyze these metrics we use gene expression programming (GEP). Here, we explore the ability of OO metrics using defect data for open source software. Further, we develop a software quality metric and suggest ways in which software professional may use this metric for process improvement. We conclude that GEP can be used in detecting fault prone classes. We also conclude that the proposed metric may be effectively used by software managers tin predicting faulty classes in earlier phases of software development.

Keywords: Metrics, Object-oriented, Software Quality, Empirical validation, Fault prediction, Gene expression programming.

1 Introduction

Faulty software classes cause software failures, increase development time, maintenance costs and decrease customer satisfaction. Effective prediction models can help software developers focus quality assurance activities on fault-prone classes and thus improve software quality by using testing resources more efficiently. Static metrics and fault data collected at class level can be used to construct fault prediction models in practice. There have been empirical studies evaluating the impact of these metrics on software quality and constructing models that utilize them in predicting quality attributes of the system, such as [1-21]. However, there is a need of data based studies to empirically validate these metrics for predicting faulty classes. In this work, we find the impact of OO metrics on fault proneness of a class using open source software Jedit [22]. We also develop a software quality metric, which can be used to predict faulty classes.

F. Bomarius et al. (Eds.): PROFES 2009, LNBIP 32, pp. 43–58, 2009.

Genetic algorithms have been successfully applied to protein structure prediction [23], defect prediction [24], and memory bound computations [25]. GEP is a type of GA as it uses population of individuals, selected them according to fitness function, and introduces genetic variation using various operators [26]. In GEP mutations insure that the resultant expression is not mathematically incorrect. "Experiments have shown that GEP is 100 to 60,000 times faster than older genetic algorithms" [27]. Thus, we build a model to predict faulty classes using the GEP. In GEP an expression is computed in order to predict faulty/non faulty classes. We analyze and validate this expression in order to predict faulty/non-faulty classes. Finally, we propose this expression as a quality metric for predicted fault prone classes. The lower values of this metric will imply higher build and release quality.

The main contributions of this study are summarized as follows: First, we empirically validated OO metrics using GEP. This method is being successfully applied in various disciplines and there is a need to evaluate its performance in predicting software quality models. Second, we used open source software system. These systems are developed with different development methods than proprietary software. In previous studies mostly proprietary software were analyzed. Third, we develop a software quality metric that can be used by software quality practitioner in earlier phases of software development to predict faulty classes. The proposed metric may also be used as quality benchmark to assess and compare software products. The results showed that the proposed metric predict faulty classes with good accuracy. However, since our analysis is based on only one data set, this study should be replicated on different data sets to generalize our findings.

The paper is organized as follows: Section 2 provides an overview of GEP. Section 3 summarizes the metrics studied, describes sources from which data is collected and gives hypothesis to be tested in the study. Section 4 presents the research methodology followed in this paper. The results of the study are given in section 5. Section 6 presents the definition and validation of the developed metric. The application of the developed software quality metric is presented in section 7. Finally, conclusions of the research are presented in section 8.

2 An Overview of Gene Expression Programming

A genetic algorithm (GA) is a search procedure with a goal to find a solution in a multidimensional space. GA is generally many times faster than exhaustive search procedures. There is a problem in finding a way to efficiently mutate and cross-breed symbolic expressions so that the resultant expressions have a valid mathematical syntax.

Candida Ferreira provided a solution to this problem [26]. Ferreira developed a system for encoding expressions so that a wide variety of mutation and cross-breeding techniques perform faster while guaranteeing that the resultant expression will always be a valid mathematical syntax. This procedure is known as GEP. GEP was presented as a new technique for the creation of computer programs. GEP uses chromosomes composed of genes organized in a head and a tail. The chromosomes are subjected to modification by means of mutation, inversion, transposition, and

recombination. The technique performs with high efficiency that greatly surpasses existing adaptive techniques.

2.1 Converting Expression Tree into k-Expression

GEP encodes the symbols in genes. This notation is called the karva language [26]. Expressions encoded using karva language are called k-expressions.

For example, the expression a+b*c can be encoded in the expression tree shown in Figure 1.

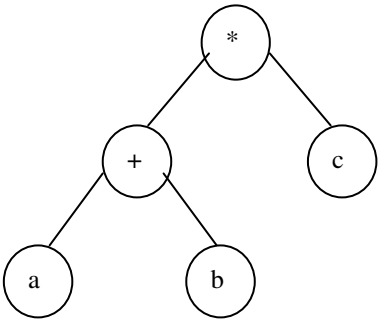

Fig. 1. Expression Tree for a+b*c

To convert the expression tree using karva language, start at the left-most symbol in top line, scan symbols from top to bottom, and left to right. The resultant k-expression is *+cab.

The process of converting an expression tree into a k-expression and vice versa can be done quickly by a computer.

2.2 Genes

The fixed number of symbols encoded in karva language constitutes a gene. A GEP gene has a head and a tail. The head can contain functions, constants, and variables whereas a tail can only contain variables and constants. The number of symbols in the head of a gene is passed as an argument in the analysis. The number of symbols in the tail is determined by the following equation

$$\text{tail} = \text{head}\ (\text{Max}-1) + 1 \tag{1}$$

where tail is the number of symbols in the tail

head is the number of symbols in the head

Max is the maximum number of operands required by any function

The tail provides a store of terminal symbols consisting of variables and constants that can be used as arguments for functions in the head. For example, head can be +,*,/ and tail can be abde. The expression is shown in Figure 2.

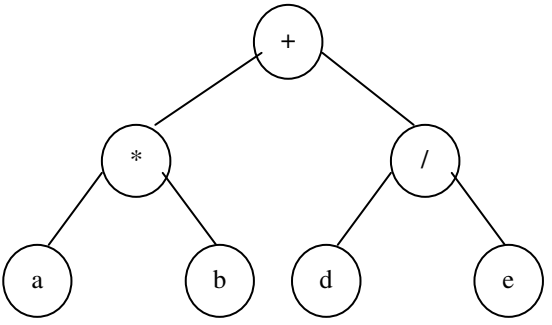

Fig. 2. Expression for Head +,*, / and Tail abde

During mutation, symbols in the head can be replaced by terminal symbols or functions whereas terminals (variables and constants) can replace symbols in the tail (see Figure 3).

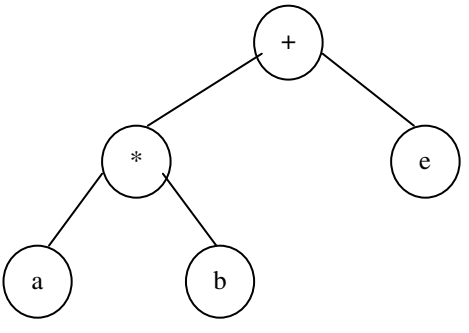

Fig. 3. Resultant Expression after Mutation

GEP ensures that the following rules are followed inorder to generate valid expression during mutation:

1. Symbols in the head are replaced with functions, constants, and variables.
2. Symbols in the tail are only replaced with variables and constants
3. The tail is of sufficient length (see equation (1))

2.3 Chromosomes

A chromosome consist of one or more than one genes of equal length. If there are more then on chromosomes in the gene, then a linking function is used to join the genes in the final function.

Consider the following example:

Gene 1: +ab
Gene 2: *cd

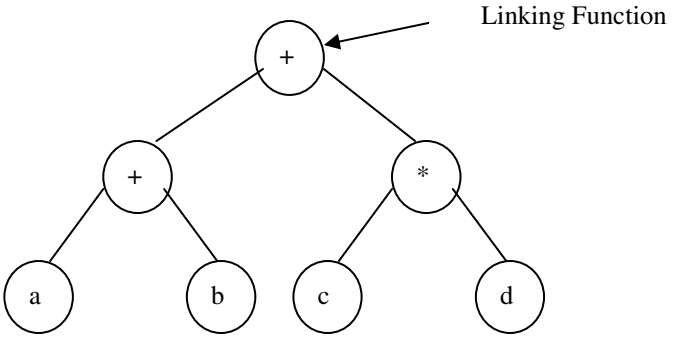

Fig. 4. Example of a 2-Gene Chromosome

The following steps are used in the training of a model using GEP:

1. Create an initial population of chromosomes.
2. Attempt to create chromosomes that model the data well.
3. Try to find a simpler function.

2.4 GEP Process

In order for a population to improve from generations to generations to predict the target variable (fault proneness in our study), mutation, inversion, transportation, and recombination are performed.

Mutation
Mutation can occur anywhere in the chromosomes but the structural organization of the chromosomes should not be changed. Mutation replaces symbols in heads of genes by function or variables and constants and symbols in tails are replaced only by variables and constants. Thus, the structural organization of chromosomes remains intact and the correct programs are produced by the mutation in the form of new individuals.

Inversion
Inversion reverses the order of symbol in a gene section.

Transposition
Transposition selects a group of symbols and moves them to a different position in the same gene.

Recombination
Two chromosomes are selected randomly and generic portion is exchanged between them inorder to produce two new chromosomes. There are three types of recombinations: one-point, two-point, and gene recombination.

3 Research Background

In this section, we present the summary of metrics studied in this paper (Section 3.1) and empirical data collection (Section 3.2).

3.1 Dependent and Independent Variables

The binary dependent variable in our study is fault proneness. The goal of our study is to empirically explore the relationship between OO metrics and fault proneness at the class level. Fault proneness is defined as the probability of fault detection in a class. We use GEP to predict probability of fault proneness. Our dependent variable will be predicted based on the faults found during software development. The software metrics [28-36] can be used in predicting these quality attributes. In this study, we empirically validated metrics given Chidamber and Kemerer [32] (see Table 1). These metrics are explained with practical applications in [28].

Table 1. Metrics Studied (Chidamber and Kemerer [32] suite)

Metric	Definition
Coupling between Objects (CBO)	CBO for a class is count of the number of other classes to which it is coupled and vice versa.
Lack of Cohesion (LCOM)	It measures the dissimilarity of methods in a class by looking at the instance variable or attributes used by methods. Consider a class C_1 with n methods M_1, M_2…., M_n. Let (I_j) = set of all instance variables used by method M_i. There are n such sets $\{I_1\}$,…….$\{I_n\}$. Let $P = \{(I_i, I_j) \mid I_i \cap I_j = 0\}$ and $Q = \{((I_i, I_j) \mid I_i \cap I_j \neq 0\}$. If all n sets $\{(I_1\}$,.....$(I_n)\}$ are 0 then P=0 $LCOM = \mid P \mid - \mid Q \mid$, if $\mid P \mid > \mid Q \mid$ $= 0$ otherwise
Number of Children (NOC)	The NOC is the number of immediate subclasses of a class in a hierarchy.
Depth of Inheritance (DIT)	The depth of a class within the inheritance hierarchy is the maximum number of steps from the class node to the root of the tree and is measured by the number of ancestor classes.
Weighted Methods per Class (WMC)	The WMC is a count of sum of complexities of all methods in a class. Consider a class K1, with methods M1,…….. Mn that are defined in the class. Let C1,…….Cn be the complexity of the methods. $WMC = \sum_{i=1}^{n} C_i$ If all method complexities are considered to be unity, then WMC = *n*, the number of methods in the class.
Response for a Class (RFC)	The response set of a class (RFC) is defined as set of methods that can be potentially executed in response to a message received by an object of that class. It is given by RFC=|RS|, where RS, the response set of the class, is given by $RS = M_i \cup_{all\ j} \{R_{ij}\}$
Number of Public Methods (NPM)	It is the count of number of public methods in a class.
Lines Of Code (LOC)	It is the count of lines in the text of the source code excluding comment lines

To incorporate the correlation of independent variables, a correlation based feature selection technique (CFS) is applied to select to select the best predictors out of independent variables in the datasets [37]. The best combinations of independent variable were searched through all possible combinations of variables. CFS evaluates the best of a subset of variables (OO metrics in our case) by considering the individual predictive ability of each feature along with the degree of redundancy between them.

3.2 Empirical Data Collection

We used JEdit open source software in this study [38]. JEdit is a programmer's text editor developed using Java language. JEdit combines the functionality of Window, Unix, and MacOS text editors. It was released as free software and the source code is available on www.sourceforge.net/projects/jedit. The LOC of JEdit is 169,107. The number of developers involved in this project was 144. The project was started in 1999.

The metric data was computed using metric tool, Understand for Java [39]. The metrics proposed by Chidamber and Kemerer [32] were computed using this tool. The number of bugs was computed using SVC repositories. The release point for the project was identified in 2002. The log data from that point to 2007 was collected. The header files in C++ were excluded in data collection. The word bug or fixed was counted. Details on bug collection process can be found in [40].

4 Research Methodology

In this section, the steps taken to analyze coupling, cohesion, inheritance and size metrics for classes taken for analysis are described. The procedure used to analyze the data collected for each measure is described in following stages (i) data statistics and outlier analysis (ii) correlation among metrics (iii) performance measures.

4.1 Descriptive Statistics and Outlier Analysis

The role of statistics is to function as a tool in analyzing research data and drawing conclusions from it. The research data must be suitably reduced so that the same can be read easily and can be used for further analysis. Descriptive statistics concern development of certain indices or measures to summarize data. The important statistics measures used for comparing different case studies include mean, median, and standard deviation. Data points, which are located in an empty part of the sample space, are called outliers. Outlier analysis is done to find data points that are over influential and removing them is essential. Univariate and multivariate outliers are found in our study. To identify multivariate outliers, we calculate for each data point the Mahalanobis Jackknife distance. Mahalanobis Jackknife is a measure of the distance in multidimensional space of each observation from the mean center of the observations [1, 41].

The influence of univariate and multivariate outliers was tested. If by removing an univariate outlier the significance (see Section 3.4) of metric changes i.e., the effect of that metric on fault proneness changes then the outlier is to be removed. Similarly, if the significance of one or more independent variables in the model depends on the presence or absence of the outlier, then that outlier is to be removed. Details on outlier analysis can be found in [42].

4.2 Correlation among Metrics

Correlation analysis studies the variation of two or more variables for determining the amount of correlation between them. In order to analyze the relationship among design metrics we use Spearman's Rho coefficient of correlation. We preferred to use a non-parametric technique (Spearman's Rho) for measuring relationship among OO metrics as we usually observe the skewed distribution of the design measures.

4.3 Evaluating the Performance of the Models

The performance of binary prediction models is typically evaluated using confusion matrix (see Table 2). In this study, we used the commonly used evaluation measures. These measures include Sensitivity, Precision, Specificity, and ROC analysis.

Table 2. Confusion matrix

Observed	Predicted	
	1.00 (Fault-Prone)	0.00 (Not Fault-Prone)
1.00 (Fault-Prone)	True Fault Prone (TFP)	False Not Fault Prone (FNFP)
0.00 (Not Fault-Prone)	False Fault Prone (FFP)	True Not Fault Prone (TNFP)

Precision
It is defined as the ratio of number of classes correctly predicted to the total number of classes.

$$\text{Precision} = \frac{TFP + TNFP}{TFP + FNFP + FFP + TNFP} \tag{2}$$

Sensitivity
It is defined as the ratio of the number of classes correctly predicted as fault prone to the total number of classes that are actually fault prone.

$$\text{Sensitivity} = \frac{TFP}{TFP + FNFP} \tag{3}$$

Sensitivity
It is defined as the ratio of the number of classes correctly predicted as not fault prone to the total number of classes that are actually not fault prone.

$$\text{Specificity} = \frac{TNFP}{FFP + FNFP} \tag{4}$$

Completeness
It is defined as the number of faults in classes classified fault-prone, divided by the total number of faults in the system.

Receiver Operating Characteristic (ROC) Analysis
ROC curve, which is defined as a plot of sensitivity on the y-coordinate versus its 1-specificity on the x coordinate, is an effective method of evaluating the quality or performance of predicted models [11]. While constructing ROC curves, one selects many cutoff points between 0 and 1 in our case, and calculates sensitivity and specificity at each cut off point. The optimal choice of cutoff point (that maximizes both sensitivity and specificity) can be selected from the ROC curve [11, 43]. Hence, by using ROC curve one can easily determine optimal cutoff point for an predicted model.

Area Under the ROC Curve (AUC) is a combined measure of sensitivity and specificity. In order to compute the accuracy of the predicted models, we use the area under ROC curve.

Cross Validation
In order to predict accuracy of model it should be applied on different data sets. We therefore performed holdout validation of models [44]. The data set is randomly divided into testing and validations data sets.

5 Analysis Results

This section presents the analysis results, following the procedure described in Section 4. Descriptive statistics (Section 5.1), GEP results (Section 5.2).

5.1 Descriptive Statistics

Table 3 show "min", "max", "mean", "std dev", "75% quartile" and "25% quartile" for all metrics considered in this study.

Table 3. Descriptive Statistics for OO metrics

Metric	Min.	Max.	Mean	Std. Dev.	Percentile (25%)	Percentile (75%)
WMC	0	407	11.72	31.201	3	10
DIT	0	7	2.496	1.976	1	3
NOC	0	35	0.715	3.100	0	0
CBO	0	105	12.64	14.13	4	17
RFC	0	843	174.97	269.5	20.75	84.25
LCOM	0	100	46.23	33.51	0	75
NPM	0	193	7.78	17.12	1	8
LOC	3	6191	206.21	529.66	32.75	171.75

The following observations are made from Table 3:

- The size of a class measured in terms of lines of source code ranges from 3-6191.
- The values of DIT and NOC are less, which shows that inheritance is not much used in all the systems; similar results have also been shown by other studies [7, 9, 10].
- The LCOM measure, which counts the number of classes with no attribute usage in common, has high values (upto 100).

We calculated the correlation among metrics as shown in Table 4 which is an important static quantity. Zhou and Leung (2006), Gyimothy, Forenc, and Siket [13] and Basili et al. [4] calculated the correlation among metrics. WMC metric is correlated with all the metrics except DIT, NOC and RFC. There is a correlation between DIT and RFC metrics, between RFC and CBO metrics, LCOM and CBO and between LCOM and NPM metrics. LOC metric is correlated with all the metrics except DIT and NOC metrics. Therefore, it shows that these metrics are not totally independent and represents redundant information.

Table 4. Correlations among Metrics

Metric	WMC	DIT	NOC	CBO	RFC	LCOM	NPM	LOC
WMC	1							
DIT	-0.17	1						
NOC	-0.005	-0.363	1					
CBO	**0.53**	0.314	-0.297	1				
RFC	0.245	**0.813**	-0.336	**0.619**	1			
LCOM	**0.632**	0.073	-0.105	**0.531**	0.340	1		
NPM	**0.822**	-0.086	-0.032	0.447	0.281	**0.570**	1	
LOC	**0.698**	0.154	-0.227	**0.841**	**0.500**	**0.620**	**0.572**	1

5.2 Gene Expression Programming (GEP) Results

In this section, we present the results of combined effect of OO metrics on fault proneness (same as multivariate analysis). The subset of attributes was selected using CFS method described in Section 3.1. NPM, CBO, RFC, DIT, and LOC were selected from the set of eight metrics.

In Table 5, we summarize the parameters to and determined by GEP. 576 generations were used to train the model to predict faulty classes and an additional generation to simplify the expression. We used 4 genes per chromosome and addition function to link the genes.

Table 5. GEP Parameters

Population size	50
Gene per chromosome	4
Gene head length	8
Generations required to train the model	576
Generations required for simplification	1
Linking Function	Addition
Fitness function	Number of correct predictions with penalty

The fitness function measures the number of correct predictions and penalties the situation where there is no correct predictions for some target categories of dependent variable.

$$Fitness = \frac{TFP + TNFP}{TFP + FNFP + FFP + TNFP}$$

If there are some correctly classified fault prone and not fault prone classes the fitness is the proportion of correctly predicted classes, but if there is no correct prediction foe either faulty or non faulty classes then the fitness is 0.

The model was applied to 274 classes and Table 6 presents the results of correctness of the fault proneness model predicted. As shown in Table 6, out of 134 classes,

Table 6. Accuracy of Model Predicted using Training Data

Observed	Predicted	
	0.00	1.00
0.00	111	29
1.00	35	99

actually fault prone, 99 classes were predicted to be fault prone. The sensitivity of the model is 73.8 percent. Similarly, 111 out of 140 classes were predicted not to be fault prone. Thus, specificity of the model is 79.28 percent. Table 7 shows the sensitivity, specificity, precision and AUC of model predicted using GEP method.

Table 7. Result of Model Training

GENE EXPRESSION PROGRAMMING	
Cutoff	0.5
Sensitivity	73.5
Specificity	79.28
Precision	76.64
AUC	0.77

6 Software Quality Metric Definition and Validation

Based on the results obtained from model prediction using GEP, we propose the generated expression as a software quality metric that can be used to predict faulty classes. The metric is defined as follows:

Metric: Fault Factor (FF)

Definition: Consider a class C_1, then the fault factor of the class is defined as follows:

$$\begin{aligned} FF &= 2*NPM*NPM + CBO + (2*(LOC+DIT)) + 2*LOC + (2*NPM) + \\ &\quad (((NPM+DIT)*NPM - LOC)) + \sqrt{RFC}) \\ &\text{if } 2*NPM*NPM + CBO + (2*(LOC+DIT)) + 2*LOC + (2*NPM) + \\ &\quad (((NPM+DIT)*NPM - LOC)) + \sqrt{RFC}) < 0, \text{ then} \\ FF &= 0 \end{aligned} \tag{5}$$

Where

NPM = Number of public methods in a class
CBO = Count of import and export coupling in a class
LOC = Lines of code in a class
DIT = Number of ancestors of a class
RFC = Number of external and internal methods in a class

When we validated the above predicted model using FF metric on 74 classes, 25 out of 38 were correctly classified as non faulty and 27 out of 36.classes were predicted to be faulty (see Table 8). Thus, the sensitivity is 75% and specificity is 65.78%. The AUC of the model is 0.704. Table 9 shows the sensitivity, specificity, precision, and AUC of model predicted using the developed metric..

Table 8. Accuracy of Model Predicted Using Validation Data

Observed	Predicted	
	0.00	1.00
0.00	25	13
1.00	9	27

Table 9. Result of Model Validation

GENE EXPRESSION PROGRAMMING	
Cutoff	0.5
Sensitivity	75
Specificity	65
Precision	69.3
AUC	0.704

7 Application of the FF Metric

Software developers can use the FF metric developed in the previous section in earlier phases of software development to measure the quality of the systems. From the design phase, one can make software measurements and then predict which classes will need extra attention during the remainder of development. The classes with higher values of FF metric will be predicted to be non faulty and the classes with less value of FF metric will be predicted as faulty. This can help management focus resources on those classes that cause most of the problems. Also, if required, developers can reconsider design and thus take corrective actions. In order to draw strong conclusions, however, more studies should evaluate the effectiveness of the proposed metric.

These design measurements can be used as quality benchmarks to assess and compare products, after one calculated the value of FF metric. More such studies can provide quality benchmarks across organizations, whereas within an organization, quality benchmarks can be set comparing metric values with the existing operational good quality software. If deviation is found in the metric values further investigation to know the cause of deviation could be done. Thus, corrective actions could be taken before final delivery or future releases of the software. This is particularly important when systems are maintained over a long period and new versions are released

regularly. Based on our observation the classes with value of FF between 2 and 392 should be classified as non faulty and the classes with values less than 2 should be classified as faulty.

Planning and resource allocating for inspection and testing is difficult. The FF metric developed in the previous section could be of great help for planning and executing testing activities. The bar chart shown in Figure 5, shows that 15.6% of classes (12 out of 134 faulty classes) misclassified as non faulty have only 1-3 number of faults. Thus, the classes with high number of faults were mostly correctly classified to be fault prone. Thus, for example, if one has the resources available to inspect 26 percent of the code. From the values calculated by the FF metric one can tell that classes with the lowest predicted metric values and total LOC upto 26% should only be tested. If these classes are selected for testing one can expect maximum faults to be covered.

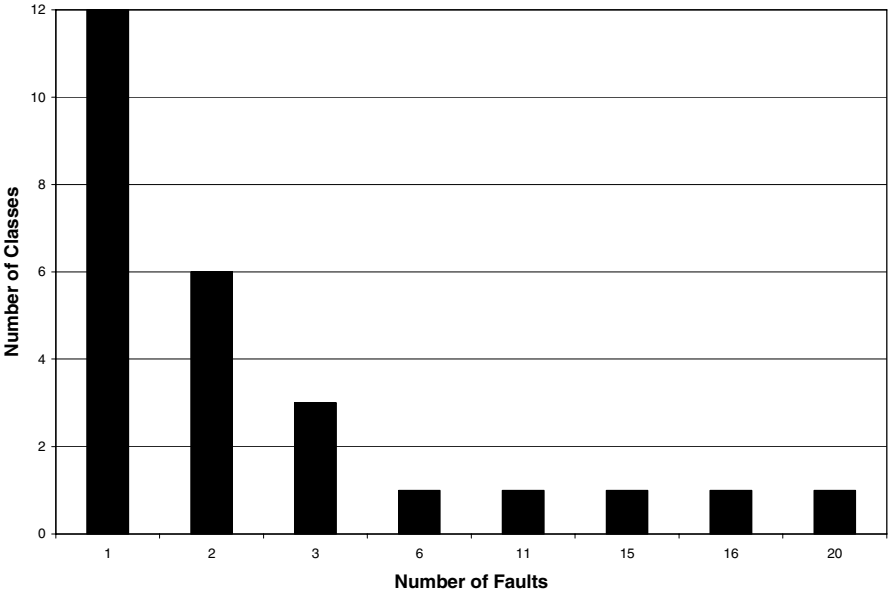

Fig. 5. Number of Faults Misclassified with Respect to Number of Classes

8 Conclusion

This paper empirically evaluates the performance of GEP algorithm in predicting fault-prone classes. We developed a software quality metric using the expression generated from GEP. The faulty classes were predicted using OO metrics proposed by Chidamber and Kemerer. The developed metric was validated using open source software. The results indicate that that the performance of GEP is at least competitive. This study confirms that construction of model using GEP is feasible, adaptable to OO systems, and useful in predicting fault prone classes.

The precision of developed metric FF is 69.3 percent, its accuracy in predicting faulty classes is 75 percent, and specificity is 65 percent. While research continues, practitioners and researchers may apply the proposed metric for predicting faulty classes. The FF metric can help in improving software quality in the context of software testing by reducing risks of faulty classes go undetected. As discussed, one important application of the proposed metric FF is to build quality benchmarks to assess fault proneness of OO systems that are newly developed or under maintenance, for example, in the case of software acquisition and outsourcing. Thus, one can conclude that FF metric appears to be well suited to develop practical quality benchmarks.

The future work may include conducting similar type of studies with different data sets to give generalized results across different organizations. We plan to replicate our study to predict model based on genetic algorithms. We will also focus on cost benefit analysis of models that will help to determine whether a given fault proneness model would be economically viable.

References

1. Aggarwal, K.K., Singh, Y., Kaur, A., Malhotra, R.: Empirical Analysis for Investigating the Effect of Object-Oriented Metrics on Fault Proneness: A Replicated Case Study. Software Process Improvement and Practice 14(1), 39–62 (2008)
2. Aggarwal, K.K., Singh, Y., Kaur, A., Malhotra, R.: Investigating the Effect of Coupling Metrics on Fault Proneness in Object-Oriented Systems. Software Quality Professional 8(4), 4–16 (2006)
3. Barnett, V., Price, T.: Outliers in Statistical Data. John Wiley & Sons, Chichester (1995)
4. Basili, V., Briand, L., Melo, W.: A Validation of Object-Oriented Design Metrics as Quality Indicators. IEEE Transactions on Software Engineering 22(10), 751–761 (1996)
5. Bieman, J., Kang, B.: Cohesion and reuse in an object-oriented system. In: Proceedings of the ACM Symposium on Software Reusability, pp. 259–262 (1995)
6. Binkley, A., Schach, S.: Validation of the coupling dependency metric as a risk predictor. In: Proceedings of the International Conference on Software Engineering, pp. 452–455 (1998)
7. Briand, L., Daly, W., Wust, J.: Exploring the relationships between design measures and software quality. Journal of Systems and Software 5, 245–273 (2000)
8. Briand, L., Wüst, J., Lounis, H.: Replicated Case Studies for Investigating Quality Factors in Object-Oriented Designs. Empirical Software Engineering: An International Journal 6(1), 11–58 (2001)
9. Cartwright, M., Shepperd, M.: An Empirical Investigation of an Object-Oriented Software System. IEEE Transactions of Software Engineering 26(8), 786–796 (1999)
10. Chidamber, S., Darcy, D., Kemerer, C.: Managerial use of Metrics for Object-Oriented Software: An Exploratory Analysis. IEEE Transactions on Software Engineering 24(8), 629–639 (1998)
11. El Emam, K., Benlarbi, S., Goel, N., Rai, S.: A Validation of Object-Oriented Metrics, Technical Report ERB-1063, NRC (1999)
12. El Emam, K., Benlarbi, S., Goel, N., Rai, S.: The Confounding Effect of Class Size on the Validity of Object-Oriented Metrics. IEEE Transactions on Software Engineering 27(7), 630–650 (2001)

13. Gyimothy, T., Ferenc, R., Siket, I.: Empirical validation of object-oriented metrics on open source software for fault prediction. IEEE Trans. Software Engineering 31(10), 897–910 (2005)
14. Harrison, R., Counsell, S.J., Nithi, R.V.: An Evaluation of MOOD set of Object-Oriented Software Metrics. IEEE Trans. Software Engineering SE-24(6), 491–496 (1998)
15. Lee, Y., Liang, B., Wu, S., Wang, F.: Measuring the Coupling and Cohesion of an Object-Oriented program based on Information flow (1995)
16. Li, W., Henry, S.: Object-Oriented Metrics that Predict Maintainability. Journal of Systems and Software 23(2), 111–122 (1993)
17. Olague, H., Etzkorn, L., Gholston, S., Quattlebaum, S.: 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 33(8), 402–419 (2007)
18. Pai, G.: Empirical analysis of Software Fault Content and Fault Proneness Using Bayesian Methods. IEEE Transactions on software Engineering 33(10), 675–686 (2007)
19. Tang, M.H., Kao, M.H., Chen, M.H.: An Empirical Study on Object-Oriented Metrics. In: Proceedings of Metrics, pp. 242–249 (1999)
20. Tegarden, D., Sheetz, S., Monarchi, D.: A software complexity model of object-oriented systems. Decision Support Systems 13(3-4), 241–262 (1995)
21. Zhou, Y., Leung, H.: Empirical analysis of Object-Oriented Design Metrics for predicting high severity faults. IEEE Transactions on Software Engineering 32(10), 771–784 (2006)
22. promise, `http://promisedata.org/repository/`
23. Moreira, B.C., Fitzjohn, P.W., Offman, M., Smith, G.R., Bates, P.A.: Novel Use of a Genetic Algorithm for Protein Structure Prediction: Searching Template and Sequence Alignment Space. PROTEINS: Structure, Function, and Genetics 53, 424–429 (2003)
24. Sheta, A.F.: Estimation of the COCOMO Model Parameters Using Genetic Algorithms for NASA Software Projects. Journal of Computer Science 2(2), 118–123 (2006)
25. Tikir, M., Carrington, L., Strohmaier, E., Snavely, A.: A Genetic Algorithms Approach to Modeling the Performance of Memory-bound Computations. In: SC 2007, Reno, Nevada, USA, November 10-16 (2007)
26. Ferreira, C.: Gene Expression Programming: A New Adaptive Algorithm for Solving Problems. Complex Systems 13, 87–129 (2001)
27. Sherrod, P.: DTreg Predictive Modeling Software (2003)
28. Aggarwal, K.K., Singh, Y., Kaur, A., Malhotra, R.: Empirical study of object-oriented metrics. Journal of Object Technology 5(8), 149–173 (2006)
29. Aggarwal, K.K., Singh, Y., Kaur, A., Malhotra, R.: Software Reuse Metrics for Object-Oriented Systems. In: Third ACIS Int'l Conference on Software Engineering Research, Management and Applications (SERA 2005), pp. 48–55. IEEE Computer Society, Los Alamitos (2005)
30. Briand, L., Daly, W., Wust, J.: Unified Framework for Cohesion Measurement in Object-Oriented Systems. Empirical Software Engineering 3, 65–117 (1998)
31. Briand, L., Daly, W., Wust, J.: A Unified Framework for Coupling Measurement in Object-Oriented Systems. IEEE Transactions on software Engineering 25, 91–121 (1999)
32. Chidamber, S., Kemerer, C.: A metrics Suite for Object-Oriented Design. IEEE Trans. Software Engineering SE-20(6), 476–493 (1994)
33. Henderson-sellers, B.: Object-Oriented Metrics, Measures of Complexity. Prentice-Hall, Englewood Cliffs (1996)
34. Hitz, M., Montazeri, B.: Measuring Coupling and Cohesion in Object-Oriented Systems. In: Proc. Int. Symposium on Applied Corporate Computing, Monterrey, Mexico (1995)

35. Lake, A., Cook, C.: Use of factor analysis to develop OOP software complexity metrics. In: Proceedings of the 6th Annual Oregon Workshop on Software Metrics, Silver Falls, Oregon (1994)
36. Lorenz, M., Kidd, J.: Object-Oriented Software Metrics. Prentice-Hall, Englewood Cliffs (1994)
37. Hall, M.: Correlation-based feature selection for discrete and numeric class machine learning. In: Proceedings of the 17th International Conference on Machine Learning, pp. 359–366 (2000)
38. jedit, http://sourceforge.net/projects/jedit/
39. scitools, http://www.scitools.com/index.php
40. Watanabe, S., Kaiya, H., Kaijiri, K.: Adapting a Fault Prediction Model to Allow Inter Language Reuse. In: PROMISE 2008, Leipzig, Germany, May 12–13 (2008)
41. Hair, J., Anderson, R., Tatham, W.: Black Multivariate Data Analysis. Pearson Education, London (2000)
42. Belsley, D., Kuh, E., Welsch, R.: Regression Diagnostics: Identifying Influential Data and Sources of Collinearity. John Wiley & Sons, Chichester (1980)
43. Hanley, J., McNeil, B.: The meaning and use of the area under a Receiver Operating Characteristic ROC curve. Radiology 143, 29–36 (1982)
44. Stone, M.: Cross-validatory choice and assessment of statistical predictions. J. Royal Stat. Soc. 36, 111–147 (1974)