ISSN 2229-5518

APPLICATION OF ARTIFICIAL NEURAL NETWORKS FOR ASSESSING THE TESTABILITY OF OBJECT ORIENTED SOFTWARE

Yogesh Singh,Anju Saha

Abstract — In this paper, we present the application of neural networks for predicting the software testability using the object oriented design metrics. The testability is generally measured in terms of the effort required for testing. The object oriented design metrics are used as the independent variables and two JUnit based test metrics are used as dependent variables in this study. The software metrics used include different measures concerning size, cohesion, coupling, inheritance, and polymorphism. This study compares the prediction performance of neural networks to the two types of statistical analysis methods: least squares regression and robust regression. This study is conducted on an agile based software, written in Java having 40K lines of code. The results of the study indicate that the prediction model using neural networks is better than that of the regression models in terms of the statistical measures of the model evaluation.

Index Terms — Artificial neural networks; Object oriented; Regression methods; Testability.

1. INTRODUCTION

The software testing is one of the costly phases of the software development life cycle. Assessment of the software testability in early stages of the software development may have a highly beneficial impact on the software testing cost and efficiency. Software testability has been defined by various researchers from different points of views. ISO defines it as "attributes of software that bear on the effort needed to validate the software product" [11]. The IEEE standard glossary defines the testability as "the degree to which a system or component facilitates the establishment of test criteria and performance of tests to determine whether those criteria have been met [10]". Voas and Miller [18] define the software testability as the probability that during testing, the software will fail on its next execution if it contains faults. Binder [2] defines software testability in terms of two properties of the software under test: controllability and observability. Bache and Mullerburg [1] define the testability in terms of the effort required for testing. Jungmayr [12] measures the testability based on the dependencies between the components. The number of dependencies increase the testing effort required to test the system. Software testability is an external software attribute which evaluates the effort and complexity required for software testing.

Most of the researchers have measured the testability in terms of the testing effort. The more the testing effort, the lesser is the testability. The software testability is a dynamic quality attribute of the software and hence it is difficult to measure it directly. However, it may be measured with the help of the static software measures. In this paper, we predict the testability of a class from the design metrics by building neural network models. We also compare the performance of neural network models with two statistical modeling techniques: least squares regression and robust regression. This study is performed upon an agile [6] based java software having 40K lines of code, which is tested using JUnit testing framework. A number of design metrics related to size, inheritance, cohesion, coupling, and polymorphism are calculated at the class level to measure the testability. The main purpose of building these models is to apply them to different kinds of systems, developed using different platforms and to focus more on the classes with lower testability.

In this paper, we make use of the two test metrics proposed by Bruntink [4] to predict the software testability. He has performed an empirical study which finds a correlation between the design metrics and the test metrics. His claim is that these test metrics provide an assessment of the testing effort which in turn provides an insight into the testability. However, Mouchawrab [16] has suggested that a multivariate model should be developed and evaluated to quantify the impact of the design metrics on the testability metrics. Our work develops the multivariate models and compares them with the neural network models.

A number of different attributes of the software quality (like maintainability, fault proneness etc.) have been

- *Yogesh Singh is currently working as vice chancellor at the Maharaja Sayaji Rao Univ Of Baroda, vadodara*
  E-mail: ys66@rediffmail.com
  Anju Saha is currently working as asst. prof. at GGSIP University, Dwarka, Delhi, E-mail: anju_kochhar@yaho.com

predicted through various empirical studies [9],[19. In these studies different statistical methods and neural networks are used to predict the attributes of the software quality. The software testability is also one of the software quality attributes which is a part of the software maintainability attribute. This study uses design metrics to model the software quality attribute (testability) using the neural networks and two statistical techniques .i.e. least squares regression and robust regression. This study is motivated by a number of factors: 1) Predicting the software testability can help in improving the software quality as it is one of the software quality attributes. 2) Software reliability can be measured from the software testability, which is one of the critical aspects of the software. 3) As software testability provides information about the testing effort required to test the system [16] it can help in planning the different testing activities. 4) The number of empirical studies are very few in the area of the software testability.

The layout of the rest of the paper is as follows. Section 2 describes the empirical study design. Section 3 describes the data analysis and research methodology. Section 4 describes the empirical results and section 5 describes the conclusions and future work.

2 THE EMPIRICAL STUDY DESIGN

In this section, we provide some background of the system that is used in this study, the data collected, the dependent and independent variables.

2.1 Description of the empirical study

This study makes use of an agile based software, "pmr", which is open source (the source code of "pmr" can be found at www.sourceforge.net) and is written in java language, with 40K lines of code. The number of java classes in "pmr" is 267. The project "pmr" is tested using the JUnit testing framework. The JUnit test classes in this project are 54.The JUnit testing framework helps to create a JUnit test class for every java class. The source code of the source java system is executed using Eclipse IDE (http://www.eclipse.org). This study is performed at the class level, hence the number of java classes used in this study are equal to the number of JUnit classes. An eclipse plugin is developed to collect the data from the above system. The values of all the design metrics are obtained for the java classes which have their JUnit classes and the value of test metrics is obtained from the corresponding JUnit classes. The values of design metrics of the java class and test metrics of the corresponding test class are paired to perform the analysis.

The project "pmr" is an open source web-based software delivery management system [sourceforge.net/projects/pmr/]. It provides a clear view of the requirements, requirements changes, project progress, risks, cost, etc. It provides tools for the team to manage tasks, quality, time, cost, iterations and deliveries. It eases communication between the project team and the customer. It monitors client satisfaction. It is suitable for any agile development processes, including XP. The project "pmr" focuses on managing customer requirements where each requirement has tasks, iterations, risks, changes, a budget, and a delivery. Some of the important features of "pmr" are: requirements and traceability management, task and iteration management, delivery management, calendar, message boards, document management, resource management, test management, bug tracking, satisfaction management, risks management, budget management etc.

2.2. Dependent variables

The aim of this study is to compare the prediction performance of the regression models and the neural network models to assess the testability of a class. We, therefore, need to select a suitable and practical measure of testability as the dependent variable for our study. In this paper, the testability is measured in terms of the testing effort of the class. The testing effort is measured through two JUnit based test metrics, which are the dependent variables for this study. The test metrics are collected from the JUnit test classes of the "pmr" system. The test metrics used in this study are: TLOC (Test Lines of Code), and TC (Number of asserts). These metrics are based at the class level and are calculated from the JUnit test classes of the java classes of the "pmr" system. TLOC and TC metrics are proposed by Bruntink and Deursen [4].

```
public class stack {
public stack()
 {
 }
public void push (int a)
          {
 }
public int pop( )
 {
  }
```

Fig 1: stack. java class

```
import
junit.framework.TestCase;
public class stackTest extends
TestCase
 {
   private stack st = new
stack();
   public void testpush()
{
   assertEquals(7, st.push(7));
 }
```

Fig 2:stackTest.java class

TLOC (Test Lines Of Code) Metric: It is defined as the number of lines of code (non comment and non blank) in a JUnit test class. In figure 2, stackTest class has TLOC = 13. Since the TLOC for stackTest class is 13.

TC (Number of asserts) Metric: It is defined as the number of asserts in the test class. In figure 2, there are two asserts

in the stackTest class. Hence for this class the value of TC= 2.

2.3 Independent Variables

The measures of size, cohesion, coupling, inheritance and polymorphism are the independent variables used in this study. More specifically, in this study, we focus on the measures defined at the class level. The object oriented metrics used in this study are given in table 1. We consider a total of 14 OO metrics, 4 size metrics, 4 coupling metrics, 3 cohesion metrics, 2 inheritance measures and one polymorphism metric. These metrics are defined in appendix A.

3 Data Analysis and Research Methodology

In this section, the methodology used to analyze the object oriented metrics and the data collected for the "pmr" system is presented. The procedure used to analyze the data collected for each metric comprises: a) The least squares linear regression model, b) the robust regression model, and c) the multivariate model using the artificial neural network to predict the testability of a class. These analysis procedures are presented in detail in the following subsections.

3.1Regression Methods

Multivariate Linear Regression (MLR) is a statistical method to model the relationship between two or more independent variables and one dependent variable which fits a linear equation to the observed data. The general form of a MLR model is given by:

Yi= a0+ a1xi1 +……. + akxin

yi= a0+ a1xi1 +…….+ akxin + ei

where xi1,…….., xin are the independent variables and a0, a1, …, an are the parameters to be estimated. Yi is the dependent variable to be predicted, yi is the actual value of the dependent variable, and ei is the error in the prediction of the ith case [17].

When building an MLR model, it is ensured that only the important independent variables are included in the resulting model through the variable selection method. There are different kinds of variable selection methods: stepwise selection, forward selection, and backward elimination. This study uses the backward elimination regression method. In the stepwise selection method, the independent variables are either added or deleted from the regression model to select an optimal subset of the independent variables for the model. In the forward selection method, the model includes the intercept at the first step. The independent variables are then included in the model depending upon some evaluation criteria, until a stopping criterion is satisfied. The backward elimination method includes all the independent variables as the first step. These variables are then removed one at a time, until a stopping criterion is satisfied.

Table 1: Metrics for Object oriented Software

S. No	Metric	OO Attribute	Sources
1	Line of code per class (LOC)	Class	[8]
2.	Number of Attributes per Class (NOA)	Class	[8]
3.	Number of Methods per Class (NOM)	Class	[8]
4.	Weighted Methods per Class (WMC)	Class	[5]
5.	Response for Class(RFC)	Coupling	[5]
6	Coupling between Objects (CBO)	Coupling	[5]
7	Data Abstraction Coupling (DAC)	Coupling	[8]
8	Message Passing Coupling (MPC)	Coupling	[8]
9.	Tight Class Cohesion (TCC)	Cohesion	[3]
10.	Information flow based Cohesion (ICH)	Cohesion	[14]
11.	Lack of Cohesion (LCOM)	Cohesion	[5]
12	Depth of Inheritance (DIT)	Inheritance	[5]
13	Number of Children (NOC)	Inheritance	[5]
14	Number of Methods Overridden by a subclass (NMO)	Polymorphism	[8]

One of the techniques of robust regression used for this study is: Least-median-squares (LMS) [15]. This is called

the robust regression method because it produces the predictive model that is more effective for making predictions for the data sets containing outliers.

The least squares regression method minimizes the sum of squared residuals as the basis for the minimization of the error; the LMS regression uses the median of the squared residuals to minimize the error.

Table 2: Descriptive statistics of pmr

Metric	Min	Max	Mean	Std. Deviation
WMC	1	45	10.93	10.248
NMO	0	2	0.44	0.839
NOA	0	7	1.76	2.037
NOM	1	28	7.28	6.965
DIT	0	2	1.35	0.520
NOC	0	2	0.06	0.302
CBO	0	15	2.13	2.882
RFC	1	62	11.69	11.472
DAC	0	7	0.89	1.355
TCC	0	1	0.41	0.428
LCOM	0	310	13.06	48.530
ICH	0	24	1.50	5.076
MPC	0	113	7.20	17.178
LOC	7	262	53.69	48.900
TLOC	3	545	119.76	136.243
TM	1	52	9.59	11.249
TC	0	121	18.98	24.574

3.2. Artificial Neural Networks

The artificial neural networks (ANN) consist of a number of interconnected neurons, each of which has a number of inputs, outputs and a transformation function. These neurons are arranged in the form of three types of layers: input layer, output layer and hidden layer(s). The input layer receives the values from the input variables and the output layer gives the output of the network [13]. Between the input and output layer there are a number of hidden layers. Each neuron is connected to another neuron through weights. The most common alternative to the MLR models is the feed-forward supervised-learning neural networks for predicting the effort and quality of the software [Zhou and Leung, (2007)]. Figure 3 shows the architecture of a feed forward neural network chosen in this study which has four layers of nodes: one input layer, one output layer and two hidden layers. Each line connecting the nodes has a unique weight associated to it. The signals in the

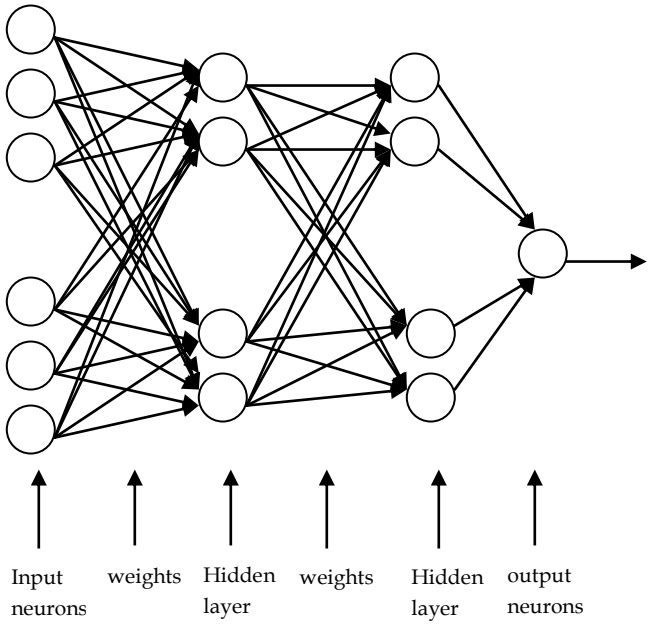

Figure 3: Architecture of the Neural Network

feed forward neural network travel in one direction only, that is from input to output. The output of each node is determined from its inputs and the weights associated with the lines between its inputs and this node. This study develops two 4-layer feed-forward back propagation networks with an input layer of fourteen nodes, 10 nodes in the first hidden layer,10 nodes in the second hidden layer, and one node in the output layer. Two networks are for two different outputs corresponding to the two dependent variables (TLOC and TC).

The input node(s) are connected to every node of the hidden layer and are not directly connected to the output nodes. All the nodes of the hidden layer are connected to the output nodes. Thus, the network does not have any shortcut connection or any lateral connection. The transfer function used between all the layers is tansig. The number of epochs used was 1000 to achieve the training goal. The network training function used is trainrp, which updates weight and bias values according to the resilient back propagation algorithm. The model is trained on approximately two thirds of the input data set and the network is tested on the remaining one third of the data set. The advantage of neural networks is that they are self-adaptive techniques and do not require more understanding with the input data. The reason for using neural networks is that the relation between the output and input variables is not

linear and is very complex.

3.3. Performance Evaluation

The "pmr" model development data set (roughly two-thirds of the entire data set) was analyzed using the spearman's correlation analyses and scatter plots, in order to identify influential variables with an aim to create a pragmatic model. The models were developed based on the results of this analysis under the least-squares regression, the robust regression methods and the neural network. The models obtained using the development data set were then applied to the testing subset (remaining one third of data set) and were evaluated using the following error measures.

1. The magnitude of relative error (MRE): The magnitude of relative error (MRE) [7] is a normalized measure of the discrepancy between the actual data values (Dact) and the predicted values (Dpred):

$$ARE = |(Dact - Dpred)| \quad (1)$$

$$MRE = ARE / Dact \quad (2)$$

The mean MRE (MMRE): MMRE [7] is the mean value for this indicator over all the observations in the data set. A software practitioner considers a model to be accurate if the value of MMRE is low.

The Pred(k) measures an indication of the overall fit for a set of data points, based on the MRE values for each data point:

$$Pred(k) = i/n \quad (3)$$

Table 3: Regression performance for TLOC

Pmr TLOC	Robust Regression			Least Squares Regression		
	All	Training	Testing	All	Training	Testing
MMRE	2.2	0.78	1.27	0.48	0.52	0.43
Std	29.3	25.68	40.53	45.9	23.1	67.3
Median	27.5	28.16	27.08	22.14	22.95	22.07
Absmean	34.12	32.89	38.42	36.81	27.15	56.9
Asum	1842.5	5141	461	1766.86	841.55	968.9
Pred(0.25)	0.33	0.48	0.33	0.39	0.42	0.353
Pred(0.30)	0.41	0.5	0.42	0.41	0.42	0.412

Pmr TC	Robust Regression			Least Squares Regression		
	All	Training	Testing	All	Training	Testing
MMRE	0.89	1.44	1.79	1.16	1.15	1.16
Std	15.5	8.06	29.17	18.9	8.2	28.82
Median	6.05	5.4	6.04	5.38	5.03	7.45
Absmean	9.91	7.31	17.57	11.9	7.85	19.32
Asum	535.3	277.89	210.8	571.7	243.3	328.43
Pred(0.25)	0.44	0.37	0.33	0.27	0.29	0.24
Pred(0.30)	0.48	0.45	0.42	0.31	0.32	0.29

Table 4: Regression performance for TC

pmr TLOC	Neural Model		
	All	Training	Testing
MMRE	0.17	0.21	0.28
Std	6.89	0.69	12.24
Median	0.56	0.229	1.32
Absmean	1.87	0.47	5.18
Asum	100.8	17.94	82.87
Pred(0.25)	0.72	0.79	0.56
Pred(0.30)	0.76	0.79	0.69

Table 5: Neural Network Performance for TLOC

pmr TC	Neural Model		
	All	Training	Testing
MMRE	0.27	0.25	0.29
Std	6.8	0.96	11.6
Median	1.03	0.64	3.3
Absmean	2.7	0.9	6.9
Asum	144.9	34.06	110.86
Pred(0.25)	0.69	0.66	0.63
Pred(0.30)	0.69	0.68	0.69

Table 6: Neural Network Performance for TC

In equation (2), k is the selected threshold value for MRE, i is the number of data points with MRE less than or equal to k, and n is the total number of data points. In this study Pred(0.25) and Pred(0.30) are used to compare the prediction performance of the models. Pred(0.25) measures the percentage of the estimates with an MRE of 25% or less, and Pred(0.3) measures the percentage of the estimates with an MRE of 30% or less. We also use the sum of AREs (Asum), the mean of AREs (Absmean), the median of AREs(Median) and the standard deviation of AREs (Std) to see the distribution of AREs. The Asum measures the ARE over the data set, the Median measures the central tendency of the distribution of ARE, the Std measures the dispersion of the distribution of ARE and the Absmean measures the mean of the AREs. These measures are used to measure the prediction performance of the models.

4 EMPIRICAL RESULTS

In this section, we analyze the results of the ANN models using the "pmr" data set. We have used MATLAB to build the back propagation trained feed-forward ANN model. To assess the benefits of using ANN, we compare the prediction performances of the ANN models with those of the MLR models. In this study, we use the backward selection procedure to build the MLR prediction models. The software used is SPSS 16.0. The entry criterion used in the backward selection is the p-value of the F statistic being smaller than or equal to 0.05 and the eliminating criterion used is the p-value of the F statistic being larger than or equal to 0.10. The software STATA is used to build the robust regression models.

4.1.Regression results

Tables 3 and 4 show the values of the performance measures achieved by both the regression models for the pmr dataset for both the dependent variables (TLOC, TC). The performance measures are calculated for all the data, the training data and the testing data. The results presented in the Tables 3 and 4 illustrate that the models built under the least-squares regression are better than the ones built with the robust regression.

Table 3 shows that the Robust regression model for TLOC has achieved the MMRE values of 2.2, 0.78 and 1.27, the pred(0.25) values of 0.33, 0.48 and 0.33 for all data, training data and testing data. Table 3 shows that the Least squares regression model for TLOC has achieved the MMRE values of 0.48, 0.52 and 0.43, the pred(0.25) value of 0.39, 0.42 and 0.353 for all data, training data and testing data. The values of MMRE are lower for the least squares regression as compared to the robust regression model.Table 4 shows that the Robust regression model for TC has achieved the MMRE values of 0.89, 1.44 and 1.79, the pred(0.25) value of 0.44, 0.37 and 0.33 for all data, training data and testing data. Table 4 shows that the Least squares regression model for TC has achieved the MMRE values of 1.16, 1.15 and 1.16, the pred(0.25) values of 0.27, 0.29 and 0.24 for all data, training data and testing data. The values of MMRE are lower for the least squares regression as compared to the robust regression model.

From the above discussion we conclude that out of the two regression models, least-squares regression model is better than the robust regression model as it has the lower value of MMRE for both the dependent variables.

4.2. The Neural Network Predictions

The two neural network models were developed for two different dependent variables (TLOC and TC) using the following steps:

1. The "pmr" data set was divided into the training and validation sets (roughly two-third and one-third respectively). The same division of data set is used for training and validation as those used in case of the regression models.
2. A number of networks were created with different network architectures and training parameters.
3. The training data set was used to train the networks, until the error is minimum.
4. The best network, in terms of having the lowest error, was used to make predictions for the testing data.
5. Different error measures are calculated on the network's performance.

The values in Tables 5 and 6 show that the neural network models for TLOC and TC perform significantly better than the corresponding regression models in Tables 3 and 4. Table 5 shows that the MMRE values for TLOC are 0.17, 0.21 and 0.28 for the neural network model, for all data, training data and testing data, respectively. Whereas the corresponding values for least squares regression are 0.48, 0.52, and 0.43 and for robust regression method are 2.2, 0.78, and 1.27. By predicting both the dependent variables the testing effort is

measured which in turn provides an insight into the testability.

The results show that the models constructed using the neural network method are feasible and adaptable to object oriented systems in order to predict the software testability. However, many more studies are needed in order to provide generalized conclusions.

5 Conclusions and Future Work

This study has compared the results of the prediction of the software testability using the ANN method with the results of the models predicted using the two types of regression methods (the least squares regression method and the robust regression method). The independent varaibles used in this work are the object oriented design metrics and the dependent varaibles are the test metrics of the object oriented software at the class level. These test metrics provide the testing effort for a class which in turn provides an assessment of the class testability. In order to compare and analyse the performance of the ANN method and the regression methods, we have used an open source software called "pmr". The values of MMRE in ANN model were small as compared to the MMRE values in regression models for both the dependent variables (TLOC and TC). Hence, the results show that the performance of the ANN method is higher than that of the regression method. Thus, ANN method may be used in predicting the software testability of object oriented systems.

However, there are a few limitations of this study. First, although the size of data set used was quite significant, further empirical studies with different systems of different sizes and different platforms are needed to strengthen the results of this study. Second, the software testability is based upon a number of aspects like the requirements, the process, and the design attributes etc. In this study, only design attributes have been used for predicting the testability. Third, this study should be extended to a number of other design level metrics which are based on other object oriented features like: polymorphism, exception handling etc.

References

[1] Bache, R.; M. Mullerburg.. "Measures of testability as a basis for quality assurance", *Software Engineering Journal*, vol.5, no.2, pp. 86-92. 1990

[2] Binder, R.V., "Design for testability in object-oriented systems", *Communication of the ACM*,vol. 37, no. 9, pp. 87–101,1994.

[3] Briand, L.; W. Daly, J. Wust., "A Unified Framework for Coupling Measurement in Object-Oriented Systems", *IEEE Transactions on software Engineering*, vol. 25, pp. 91-121,1999.

[4] Bruntink, M..;A.V. Deursen., "An Empirical Study into Class Testability", *Journal of systems and software* vol ,79, no. 9, pp.1219–1232,2006.

[5] Chidamber, S., C. Kemerer, "A metrics suite for object oriented design", *IEEE Transactions on Software Engineering*, vol 20, no. 6, pp. 476–493,1994.

[6] Cockburn, A., *Agile Software Development*, Addison-Wesley, 2002.

[7] Gray, A. R.., MacDonell, S. G., "Software metrics data analysis: exploring the relative performance of some commonly used modeling techniques", *Empirical Software Engineering*, vol. 4, pp. 297–316, 1999.

[8] Henderson-Sellers, B., *Object-Oriented Metrics*, Prentice Hall, 1996.

[9] Heung., Seok Chae., Tae Yeon Kim Woo., Sung Jung., Joon-Sang Lee., "Using Metrics for Estimating Maintainability of Web Applications: An Empirical Study", *6th IEEE/ACIS International Conference on Computer and Information Science*, pp. 1053 – 1059, 2007.

[10] IEEE Press, IEEE Standard Glossary of Software Engineering Technology, ANSI/IEEE Standard, 1990.

[11] ISO, International standard ISO/IEC 9126", Information technology: Software Product Evaluation: Quality Characteristics and Guidelines for their Use, 1991

[12] Jungmayr, S., "Identifying Test-Critical Dependencies", *Proceedings of the IEEE International Conference on Software Maintenance*, pp. 404–413, 2002.

[13] Khoshgoftaar, T.M., David L. Lanning, "A Neural Network Approach for Early Detection of Program Modules Having High Risk in the Maintenance Phase", *Journal of Systems and Software*, vol. 29, pp. 85-91, 1995.

[14] Lee.Y., B.Liang., S.Wu., F.Wang., "Measuring the Coupling and Cohesion of an Object-Oriented program based on Information flow", *In proceedings of the international conference on software quality*, 1995.

[15] Rousseeuw, P. J.; Leroy, A. M., *Robust Regression and Outlier Detection*, John Wiley & Sons, New York NY, USA, 1987.

[16] S., Lionel C., Briand, Yvan Labiche., "A measurement framework for object-oriented software testability", *Information and Software Technology*, vol. 47, pp.979–997, 2005.

[17] T. Khoshgoftaar., M. Seliya, "Fault prediction modeling for software quality estimation: comparing commonly used techniques", *Empirical Software Engineering*, vol. 8, no. 3, pp. 255–283, 2003.

[18] Voas, J.M., K.W. Miller, "Software testability: the new verification", *IEEE Software*, vol. 12, no. 3, pp. 17–28, 1995.

[19] Yogesh Singh., Bindu Goel, "An Integrated Model to Predict Fault Proneness Using Neural Networks", *Software Quality Professional*, vol. 10, no. 2, pp. 22-32, 2008.

[20] Yuming Zhou.,Hareton Leung, "Predicting object-oriented software maintainability using multivariate adaptive regression splines", *Journal of Systems and Software*, vol. 80, pp. 1349-1361, 2007.

APPENDIX

Size metrics

In this section four size metrics are discussed. These metrics measure the size of the system in terms of lines of code, attributes and methods included in the class. As these metrics capture the complexity of the class hence they can give an insight into the testability of the class.

Lines of code per class (LOC)
It counts the total number of lines of code (non-blank and non-comment lines) in the class.
Number of Attributes per Class (NOA)
It counts the total number of attributes defined in a class.
Number of Methods per Class (NOM)
It counts number of methods defined in a class.
Weighted Methods per Class (WMC)
The WMC is the count of the sum of the McCabe's Cyclomatic Complexity for all the methods in the class. If method complexity is one for all the methods, then WMC = n, the number of methods in the class.

Cohesion Metrics

Cohesion measures the degree to which the elements of a module are functionally related. A strongly cohesive module does little or no interaction with other modules and implements the functionality which is related to only one feature of the software. This study considers following three cohesion metrics.

Lack of Cohesion in Methods (LCOM)
Lack of Cohesion (LCOM) measures the cohesiveness of the class. It is defined as below:
Let M be the set of methods and A be the set of attributes defined in the class. Ma is the number of methods that access a. Mean be the mean of Ma over A.Then,
LCOM = (Mean -|M|)/(1 - |M|)
Information flow based Cohesion (ICH)
ICH for a class is defined as the number of invocations of other methods of the same class, weighted by the number of parameters of the invoked method.
Tight Class Cohesion (TCC)
The measure TCC is defined as the percentage of pairs of public methods of the class with common attribute usage.

Coupling metrics
Coupling relations increase complexity and reduce encapsulation.

Coupling Between Objects (CBO)
CBO for a class is the count of the number of other classes to which it is coupled. Two classes are coupled when methods declared in one class use methods or instance variables defined by the other class.

Data Abstraction Coupling (DAC)
Data Abstraction is a technique of creating new data types suited for an application to be programmed. Data Abstraction Coupling (DAC) is defined as the number of ADTs defined in a class.
Message passing Coupling (MPC)
Message Passing Coupling (MPC) is defined as the "number of send statements defined in a class". So if two different methods in class C access the same method in class D, then MPC = 2.
Response for a Class (RFC)
The response for a class (RFC) is defined as the set of methods that can be executed in response to a message received by an object of that class.

Inheritance Metrics
This section discusses two different inheritance metrics, which are considered for this study.

Depth of Inheritance Tree (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.
Number of Children (NOC)
The NOC is the number of immediate children of a class in the class hierarchy.

Polymorphism Metrics
Polymorphism is the characteristic of object oriented software through which the implementation of a given operation depends on the object that contains the operation.

Number of Methods Overridden by a subclass (NMO)
When a method in a child class has the same name and signature as in its parent class, then the method in the parent class is said to be overridden by the method in the child class.