# Predicting Object-Oriented Software Maintainability using Hybrid Neural Network with Parallel Computing Concept

Lov Kumar[1]
Dept. CS&E
NIT Rourkela
lovkumar505@gmail.com

Santanu Ku. Rath[2]
Dept. CS&E
NIT Rourkela
skrath@nitrkl.ac.in

## ABSTRACT

Software maintenance is an important aspect of software life cycle development, hence prior estimation of effort for maintainability plays a vital role. Existing approaches for maintainability estimation are mostly based on regression analysis and neural network approaches. It is observed that numerous software metrics are even used as input for estimation. In this study, Object-Oriented software metrics are considered to provide requisite input data for designing a model. It helps in estimating the maintainability of Object-Oriented software. Models for estimating maintainability are designed using the parallel computing concept of Neuro-Genetic algorithm (hybrid approach of neural network and genetic algorithm). This technique is employed to estimate the software maintainability of two case studies such as the User Interface System (UIMS), and Quality Evaluation System (QUES). This paper also focuses on the effectiveness of feature reduction techniques such as rough set analysis (RSA) and principal component analysis (PCA). The results show that, RSA and PCA obtained better results for UIMS and QUES respectively. Further, it observed the parallel computing concept is helpful in accelerating the training procedure of the neural network model.

## Categories and Subject Descriptors

D.1.5 [**PROGRAMMING TECHNIQUES**]: Object-oriented Programming; D.2.8 [**Software Engineering**]: Metrics; I.2.6 [**ARTIFICIAL INTELLIGENCE**]: Learning

## Keywords

Artificial neural network, Maintainability, Genetics algorithm, Object-Oriented Metrics, Parallel Computing

## 1. INTRODUCTION

Present day emphasis is mostly given on Object-Oriented paradigm for software development. The quality of Object-Oriented software is assessed by the use of software metrics. A number of metrics have been proposed for this purpose. Some of the metrics available in literature are as follows: Abreu MOOD metric suite [1], Bieman and Kang [14], Briand *et al.* [4], Halstead [11], Henderson-sellers [12], Li and Henry [19], McCabe [21], Lorenz and Kidd [20], Lake and Cook [18] and CK metric [7] suite, etc.

The usefulness of these metrics lies in their ability to predict the quality of the developed software. Software quality attributes, identified by ISO/IEC 9126 are functionality, reliability, usability, maintainability, portability and efficiency [13]. In this study, focus has been given on a particular attribute i.e., maintainability in order to improve design or coding of Object-Oriented software. It also provide some useful information for plan the use of valuable resources [28]. The ISO/IEC 9126 standard defines maintainability as the capability of the software product to be modified, including corrections, improvements or adaptation of the software to changes in environment and in requirements and functional specifications. In this paper, maintainability is considered as the number of source of lines changed per class [13]. A line change can be an 'addition' or 'deletion' of lines of code in a class [19].

In order to estimate the maintainability of a class, several traditional methods are available in literature as proposed by many authors. But less importance has been given on using machine learning techniques for estimation purpose. Artificial intelligence techniques, a subset of machine learning methods have the ability of computer, software and firmware to measure the properties of a class, that human beings recognize as intelligent behavior. These methods are able to approximate the non-linear function with more precision. Hence they can be applied for software maintainability estimation in order to achieve better accuracy. In this paper, hybrid approach of ANN and genetic algorithm i.e., Neuro-genetic (Neuro-GA) [5] approach with parallel computing framework is used for predicting software maintainability on two commercial software products such as User Interface System (UIMS) and Quality Evaluation System (QUES). To train these models, Object-Oriented software metrics are considered as input data.

The remainder of the paper is organized as follows: Section 2 shows the related work in the field of software maintainability estimation and Object-Oriented metrics. Section 3 emphasizes on mining of metrics values from data repository. Section 4 briefs about the methodologies used to estimate the maintainability. Section 5 shows the concept of parallel computing for training the neural network. Section 6 presents the performance parameters used for evaluating the


*ISEC* '15 Bengaluru, Karnataka, India

http://dx.doi.org/10.1145/2723742.2723752.

models. Section 7 highlights on the results for maintainability prediction, achieved by applying Neuro-GA approach. Section 8 gives a note (comparison) on the performance of the designed models based on the performance parameters. Section 9 concludes the paper with scope for future work.

## 2. RELATED WORK

It is observed in literature that software metrics are used in design of prediction models which serve the purpose of computing the prediction rate in terms of accuracy such as fault, effort, re-work and maintainability. In this paper, emphasis is given on work done on the use of software metrics for maintainability prediction.

Table 1 shows the summary of literature review done on maintainability, where it describes the applicability of numerous software metrics used by various researchers and practitioners in designing their respective prediction models.

**Table 1: Summary of Empirical Literature on Maintainability**

| Author | Prediction technique |
|---|---|
| Li and Henry (1993) [19] | Regression based models |
| Paul Oman (1994) *et al.* [24] | Regression based models |
| Don Coleman (1994) *et al.* [8] | Hierarchical multidimensional assessment model, polynomial regression models, aggregate complexity measure, principal components analysis, and Factor analysis. |
| Don Coleman (1995) *et al.* [9] | Hierarchical multidimensional assessment model, polynomial regression models, Estimating maintainability via entropy, principal components analysis, and Factor analysis. |
| Scott L. Schneberger (1997) [26] | Regression based models |
| Binkley *et al.* (1998) [3] | Regression analysis |
| Bandi *et al.* (2003) [2] | variance, correlation, and regression analysis. |
| Van Koten *et al.* (2006) [27] | Bayesian Network, Regression Tree , Backward Elimination and Stepwise Selection. |
| Yuming Zhou and Hareton Leung (2007) [28] | Multivariate linear regression, Artificial neural network, Regression tree, Support vector regression and Multivariate adaptive regression splines |
| Jie-Cherng Chen and Sun-Jen Huang (2009) [6] | Regression analysis |

From the above table, it can be interpreted that many of the authors have used statistical methods such as regression based analysis and their forms in predicting the maintainability. But keen observation reveals that very less work has been carried out on using neural network models for designing their respective prediction models.

Neural network models over the years have seen an explosion of interest, and their applicability across a wide range of problem domains. Indeed, neural network models can be mainly used to solve problems related to prediction and classification. Neural network models act as efficient predictors of dependent and independent variables due to sophisticated modeling technique where in, they posses the ability to model complex functions. In this paper, software metrics have been considered for predicting maintainability by applying artificial intelligence techniques.

## 3. RESEARCH BACKGROUND

The following subsections highlight on the data set used for computing maintainability. Data are normalized to obtain better accuracy and then dependent and independent variables are chosen for maintainability estimation.

### 3.1 Metrics set and empirical data collection

Metrics suites are defined for different goals such as effort estimation, fault prediction, re-usability and maintenance effort. In this paper, different Object-Oriented metrics have been considered for predicting software maintainability i.e., the number of lines changed per class is considered as a criterion in determining the maintainability of a class. A line change can be an 'addition' or 'deletion' of lines of code in a class [19]. The metrics selected in this study are tabulated in Table 2. Classic-ADa is an Object-Oriented programing language that adds the capability of Object-Oriented programing to ADa by providing Object-Oriented construct in addition to the ADa constructs [19]. In this paper, Classic-ADa metrics analyzer is used to gather metrics from Classic-ADa's design and source code. Figure 1 depicts the flow chart to extract the *change* i.e., altered lines of code in the developed software.

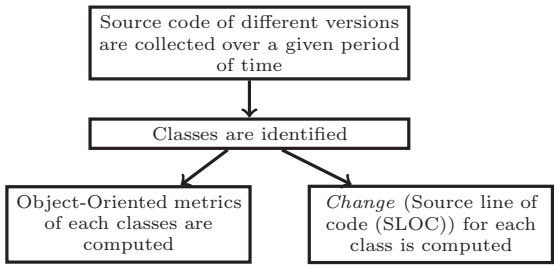


**Figure 1: Flow chart for extraction of *change* in a software product**

### 3.2 Effectiveness of metrics

After obtaining the maintainability data, an attempt is made to establish a relationship between the maintainability and the metrics. The maintainability 'change' is measured as "the number of lines changed per class' [19]. Hence in this approach, change is considered as a dependent variable and each of the metric group as a set of independent variables while developing the relation. Maintainability is thus assumed to be a function of the used metrics.

To analyze the effectiveness of the metrics used, in the paper the various metrics are categorized into different groups as follows:

a. **Analysis 1 (A1)**: The effectiveness of SIZE metrics

**Table 2: Definition of the metrics used**

| Metric | Description |
|---|---|
| Weighted method per class (WMC) | Sum of the complexities of all class methods. |
| Depth of inheritance tree (DIT) | Maximum length from the node to the root of the tree. |
| Number of children (NOC) | Number of immediate sub-classes subordinate to a class in the class hierarchy. |
| Response for class (RFC) | A set of methods that can potentially be executed in response to a message received by an object of that class. |
| Lack of cohesion among methods (LCOM) | Measures the dissimilarity of methods in a class via instanced variables. |
| Data abstraction coupling (DAC) | The number of abstract data types defined in a given class. |
| Message-passing coupling (MPC) | The number of send statements defined in a given class. |
| Number of methods (NOM ) | The number of methods implemented within a given class. |
| SIZE1 | The number of semicolons in a given class |
| SIZE2 | Total number of attributes and the number of local methods in a given class. |

**Table 3: Descriptive statistics of classes for UIMS**

| UIMS | WMC | DIT | NOC | RFC | LCOM | MPC | DAC | NOM | SIZE1 | SIZE2 | CHANGE |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Max. | 69 | 4 | 8 | 101 | 31 | 12 | 21 | 40 | 439 | 61 | 289 |
| Min. | 0 | 0 | 0 | 2 | 1 | 1 | 0 | 1 | 4 | 1 | 2 |
| Median | 5 | 2 | 0 | 17 | 6 | 3 | 1 | 7 | 74 | 9 | 18 |
| Mean | 11.38 | 2.15 | 0.94 | 23.20 | 7.48 | 4.33 | 2.41 | 11.38 | 106.44 | 13.97 | 46.82 |
| Std Dev. | 15.89 | 0.90 | 2.01 | 20.18 | 6.10 | 3.41 | 4.00 | 10.21 | 114.65 | 13.47 | 71.89 |

**Table 4: Descriptive statistics of classes for QUES**

| QUES | WMC | DIT | NOC | RFC | LCOM | MPC | DAC | NOM | SIZE1 | SIZE2 | CHANGE |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Max. | 83 | 4 | 0 | 156 | 33 | 42 | 25 | 57 | 1009 | 82 | 42.09 |
| Min. | 1 | 0 | 0 | 17 | 3 | 2 | 0 | 4 | 115 | 4 | 6 |
| Median | 9 | 2 | NA | 40 | 5 | 17 | 2 | 6 | 211 | 10 | 52 |
| Mean | 14.95 | 1.91 | 0 | 54.38 | 9.18 | 17.75 | 3.44 | 13.41 | 275.58 | 18.03 | 62.18 |
| Std Dev. | 17.05 | 0.52 | 0 | 32.67 | 7.30 | 8.33 | 3.91 | 12.00 | 171.60 | 15.21 | 42.09 |

along with the combination of CK metrics suite and Li and Henry metrics are used for estimating the maintainability. The relationship is represented as follows:

$$Maintainability = Change = f(WMC, DIT, NOC, RFC, LCOM, MPC, DAC, NOM, SIZE1, SIZE2)$$

b. **Analysis 2 (A2)**: In this analysis, feature extracted attributes using principal component analysis (PCA) are considered as input for estimating the maintainability. The relationship is represented as follows:

$$Maintainability = Change = f(\text{ Extracted feature attributes using PCA})$$

c. **Analysis 3 (A3)**: In this analysis, reduced feature attributes using rough set analysis (RSA) are considered as input to design a model for estimating the maintainability. The relationship is represented as follows:

$$Maintainability = Change = f(\text{ Reduced feature attributes using RSA })$$

## 3.3 Case study

In this paper, to analyze the effectiveness of the proposed approach, two Object-Oriented software data sets published by Li and Henry (1993) are used as case studies [19]. Softwares such as User Interface System (UIMS) and Quality Evaluation System (QUES) are chosen for computing the maintainability. The softwares systems viz., UIMS and QUES have 39 and 71 classes respectively. The data have been collected over the past three years. The maintainability of a software is measured by the number of lines changed per class. Table 3 and Table 4 show the Min, Max, Median and Standard deviation values of the two software systems (UIMS and QUES).

In this analysis, the derivative of inheritance metric 'NOC' in QUES software product, has all its 71 classes with NOC values *zero*. This indicates that there are no immediate subclasses of a class in the class hierarchy and hence NOC is not considered in computing maintainability in this analysis. From Table 3 and Table 4, it is clear that the DIT metric has low value of median and mean for both UIMS and QUES data sets. The low value of median and mean for DIT shows that inheritance was considered to a greater extent in both software system. Similarly medians and means of NOM and SIZE2 are found in the UIMS and QUES data sets, suggesting that the class size at the design level in both systems are similar. However, the medians and means of SIZE1 in the QUES data set are significantly larger than those in the UIMS data set. This suggests that the complexities of the problems processed by the two systems are rather different. Moreover, the medians and means of RFC and MPC in the QUES data set are of greater value in comparison with UIMS data set. This suggests that the coupling between classes in

**Table 5: Correlations between the metrics for UIMS (upper triangle) and QUES (lower triangle)**

| | WMC | DIT | NOC | RFC | LCOM | MPC | DAC | NOM | SIZE1 | SIZE2 | CHANGE |
|---|---|---|---|---|---|---|---|---|---|---|---|
| WMC | 1 | -0.22 | 0.23 | 0.91 | 0.80 | 0.63 | 0.44 | 0.84 | 0.97 | 0.77 | 0.65 |
| DIT | -0.13 | 1 | -0.47 | -0.23 | -0.19 | 0.06 | -0.43 | -0.36 | -0.19 | -0.41 | -0.43 |
| NOC | NA | NA | 1 | 0.21 | 0.13 | 0.03 | 0.32 | 0.23 | 0.17 | 0.27 | 0.56 |
| RFC | 0.74 | 0.11 | NA | 1 | 0.79 | 0.74 | 0.61 | 0.93 | 0.91 | 0.89 | 0.64 |
| LCOM | 0.57 | 0.12 | NA | 0.82 | 1 | 0.50 | 0.36 | 0.75 | 0.82 | 0.68 | 0.57 |
| MPC | 0.14 | 0.02 | NA | 0.33 | -0.10 | 1 | 0.44 | 0.55 | 0.67 | 0.55 | 0.45 |
| DAC | 0.57 | 0.39 | NA | 0.64 | 0.56 | 0.02 | 1 | 0.75 | 0.52 | 0.87 | 0.63 |
| NOM | 0.70 | 0.13 | NA | 0.81 | 0.88 | -0.11 | 0.81 | 1 | 0.87 | 0.98 | 0.64 |
| SIZE1 | 0.89 | 0.01 | NA | 0.80 | 0.54 | 0.37 | 0.64 | 0.69 | 1 | 0.82 | 0.63 |
| SIZE2 | 0.69 | 0.20 | NA | 0.81 | 0.84 | -0.08 | 0.89 | 0.99 | 0.71 | 1 | 0.67 |
| CHANGE | 0.43 | -0.09 | NA | 0.39 | 0.05 | 0.46 | 0.08 | 0.14 | 0.64 | 0.15 | 1 |

the QUES is higher than those in the UIMS. In contrast, the median and mean of LCOM in the QUES data set are similar to the median and mean of LCOM in the UIMS data set, implying that these two systems have similarly cohesion. It can also be seen that the mean of CHANGE in the QUES data set is larger than that in the UIMS data set. The dependency between metrics is computed using *Pearson's* correlations ($r$: Coefficient of correlation) for UIMS and QUES. The coefficient of correlation, $r$, is useful because it measures the strength and direction of the linear relationship between two variables. It is defined as the covariance of the variables divided by the product of their standard deviations. It also act as a measure that allows us to determine how certain one can be in making predictions from a certain model. Table 5 shows the *Pearson's* correlation analysis for the dataset. The upper triangular matrix represents the correlations between the metrics in the UIMS data set, and the lower triangular matrix represents the correlations between the metrics in the QUES data sets.

## 3.4 Data normalization technique

Normalization of input feature values has been carried out, over the range [0,1], so as to adjust the defined range of input feature values and avoid the saturation of neurons, when we apply for neural network. In this paper, Min-Max normalization technique has been used to normalize the data [15].

Min-Max normalization performs a linear transformation on the original data. It maps each of the actual data $x$ of attribute $X$ to normalized value x′ which lie in the range of [0,1]. Min-Max normalization is calculated by using the following Equation:

$$Normalized(x) = x' = \frac{x - min(X)}{max(X) - min(X)} \qquad (1)$$

where min(X) and max(X) represent the minimum and maximum values of the attribute X respectively.

## 3.5 Cross-validation method

Cross-validation is a statistical learning method which is used to evaluate and compare the models by partitioning the data into two portions. One portion of the divided set is used to train or learn the model and the rest of the data is used to validate the model.

K-fold cross-validation is the basic form of cross validation [17]. In K-fold cross-validation the data are first partitioned into $K$ equal (or nearly equally) sized portions or folds. For each of the $K$ model, $K$-1 folds are used for training and the remaining one fold is used for testing purpose. The significance of K-fold-cross-validation lies in it's ability to use the data set for both training and testing. So the performance of each model on each fold can be tracked using predetermined performance metrics available in literature. In literature, it is observed that 5-fold and 10-fold cross-validation approaches have been used for designing a model. In this paper, 5-fold cross-validation are used for both QUES and UIMS for comparing the models.

## 3.6 Principal Component Analysis (PCA)

The concept of principal component analysis (PCA) was develop by Karl Pearson in 1901. PCA is a statistical technique used to transfer a data space of high dimension into a feature space of lower dimension having the most significant features. PCA rigidly rotates the axes of the p-dimension space to new position (principle axes) such that principle axis 1 has the highest variance, axis 2 has the next highest variance and so on.

Many of the Object-Oriented have high correlation with each other. So PCA is used to transfer raw metrics to variables that are not correlated to each other when the original data are Object-Oriented metrics, called a new principal component variables domain metrics. The detail steps of PCA is describe below:

***PCA()***

**Input**: 'n x m' feature matrix $X$ where 'n' represents number of samples and 'm' represents the number of features.
**Output**: 'nxk' reduced feature matrix ($k << m$).

***Step 1:*** Matrix 'X' is normalized ensure zero mean of each feature value.

Evaluate $\mu_j = \frac{1}{n}\sum_{i=1}^{n} x_i^j$ vary j for all feature values i.e., 1 to m

Replace $x^j$ with $(x^j - \mu_j)$ vary $x^j$ across all samples i.e., from 1 to n

***Step 2:*** Compute covariance matrix of the normalized matrix.

$\sum(sigma) = \frac{1}{m}(X^T X)$

***Step 3:*** Compute the eigen vectors of matrix using MATLAB command as: $eign = eig(sigma)$

***Step 4:*** Choose the first 'k' number of principal components from the covariance matrix using the following criteria:

**for** (*every eigen vector* $i = 1$ *to* $m$ ) **do**

Evaluate $cumvar = \frac{\sum_{i=1}^{k} \lambda_{ii}}{\sum_{i=1}^{m} \lambda_{ii}}$ {cumvar denotes (cumulative variance) and ($\lambda$) represents eigen values sorted in descending order}

**if** $(cumvar \geq 0.99) or (1 - cumvar \leq 0.01)$ **then**

return k {99% of variance is retained}

**end if**

**end for**

***Step 5:*** Reduce the matrix dimension, taking the first k columns (1 to k) of eign matrix as eign(:,1:k) and assign to $eign_{red}$.

***Step 6:*** Evaluate $Z = X * eign_{red}$.

where Z is the new matrix with reduced feature dimension retaining 99% of the variance.

***Step 7:*** Stop.

## 3.7 Rough Set Analysis

Rough set analysis (RSA) is a formal approximation of a conventional (CRISP) set, which was described by Pawlak [25]. This formal approximation, represents the lower and upper bound of the original set. It helps in adequate analysis of various types of data, especially when dealing with inexact, vague and uncertain data. Rough set captures two unique features of imperfection in knowledge i.e., in-discernibility and vagueness. Rough set execution is based on the concept that lowering the 'degree of precision' in the data makes data pattern more visible. In general, rough set approach can be viewed as a formal framework for mining facts from imperfect data. The results achieved by application of rough set concept can be represented in the form of classification, decision rules or inform of reduced data set. Following are the steps followed to obtain reduced attribute set:

Step 1. Collection of data.
Data is extracted from Promise data repository [[22]].

Step 2. Discretization of data.
The data extracted from the repository is discretized by using K-means clustering algorithm.

Step 3. Lower and upper approximation of all possible set is calculated.
Lower approximation is defined as the union of all these elementary sets which are contained in X.

$$\underline{B}X = \{x_i \in U | \ [x_i]_{Ind(B)} \subset X\} \tag{2}$$

Upper approximation is the union of these elementary sets, which have a non-empty intersection with X.

$$\bar{B}X = \{x_i \in U | \ [x_i]_{Ind(B)} \cap X \neq 0\} \tag{3}$$

Step 4. Accuracy of all possible set is calculated.
An accuracy measure of the set X in B $\subseteq$ A is defined as:

$$\mu_B = \frac{Card(\underline{B}X)}{Card(\bar{B}X)} \tag{4}$$

The cardinality of a set is the number of objects contained in the lower / upper approximation of the set X.

Step 5. All possible sets are selected on the basis such that, their accuracy is equal to the accuracy of universal set.

Step 6. The set with least possible value of cardinality is chosen as reduct set from all possible selected set.

# 4. PROPOSED WORK FOR PREDICTING MAINTAINABILITY

This section gives a brief note on the use of a hybrid approach on Artificial neural network (ANN) and genetic algorithm (GA) i.e.,Neuro-GA approach for predicting software maintainability [5].

## 4.1 Neuro-GA Approach

In this approach, genetic algorithm is used for updating the weight during learning phase. A neural network with a configuration of 'l-m-n' is considered for estimation i.e., the network consists of 'l' number of input neurons, 'm' number of hidden neurons, and 'n' number of output neurons. In this paper, for input layer, linear activation function is used i.e., the output of the input layer is treated as input of the input layer. It is represented as:

$$O_i = I_i \tag{5}$$

For hidden layer and output layer, sigmoidal function or squashed-S function is used. The output of hidden layer '$O'_h$ for input of hidden layer '$I'_h$ is represented as:

$$O_h = \frac{1}{1 + e^{-I_h}} \tag{6}$$

and output of the output layer '$O'_o$ for the input of the output layer '$I'_o$ is represented as:

$$O_o = \frac{1}{1 + e^{-I_o}} \tag{7}$$

The number of weights $N$ required for this network with a configuration of 'l-m-n' can be computed using the following equation:

$$N = (l + n) * m \tag{8}$$

Each weight (gene) being a real number and assuming the number of digits (gene length) in weights to be $d$, the length of the chromosome $L$ can be computed using the following equation:

$$L = N * d = (l + n) * m * d \tag{9}$$

For determining the fitness value of each chromosome, weights are extracted from each chromosome using the following equation:

$$W_k = \begin{cases} \text{if } 0 <= x_{kd+1} < 5 \\ -\frac{x_{kd+2} * 10^{d-2} + x_{kd+3} * 10^{d-3} + \ldots + x_{(k+1)d}}{10^{d-2}} \\ \text{if } 5 <= x_{kd+1} <= 9 \\ +\frac{x_{kd+2} * 10^{d-2} + x_{kd+3} * 10^{d-3} + \ldots + x_{(k+1)d}}{10^{d-2}} \end{cases} \tag{10}$$

The fitness value of each chromosome is determined based on the derived fitness function. The algorithm for deriving fitness function is as follows:

Let $(\bar{I}_i, \bar{T}_i)$ ; i=1,2,3....,N where

$\bar{I}_i = (I_{1i}, I_{2i}, I_{3i}, ...., I_{li})$ and $\bar{T}_i = (T_{1i}, T_{2i}, T_{3i}, ...., T_{ni})$

represent the respective input and output pairs of the neural network with a configuration of l-m-n. For each chromosome $C_i, i = 1, 2, 3, ...., p$, belonging to the current population $P_i$ whose size is $P$. The following algorithm indicates the steps to find the fitness value of the individual chromosomes in the population:

***Algorithm for fitness function: FITGEN()***

**Input**: $\bar{I}_i = (I_{1i}, I_{2i}, I_{3i}, ...., I_{li})$
**Output**: $\bar{T}_i = (T_{1i}, T_{2i}, T_{3i}, ...., T_{ni})$
where $\bar{I}_i$, $\bar{T}_i$ represent the input and output pairs of the l-m-n configuration of neural network.

***Step 1:*** Weights $\bar{W}_i$ from $C_i$ are calculated using equation 10.
***Step 2:*** Considering $\bar{W}_i$ as a constant weight, the network is trained for $N$ input instances and the estimate value $O_i$ is found.
***Step 3:*** Error $E_j$ for each input instance $j$ is computed using following equation:

$$E_j = (T_{ji} - O_{ji})^2 \tag{11}$$

***Step 4:*** Root mean square error (RMSE) for the chromosome $C_i$ is computed using the following equation:

$$E_i = \sqrt{\frac{\sum_{j=1}^{j=N} E_j}{N}} \tag{12}$$

where $N$ is the total number of training data set.
***Step 5:*** Fitness value for chromosome $C_i$ using the following equation is found out as:

$$F_i = \frac{1}{E_i} = \frac{1}{\sqrt{\frac{\sum_{j=1}^{j=N} E_j}{N}}} \tag{13}$$

Figure 2 shows the block diagram for Neuro-GA approach, which represents the steps followed to design the model.

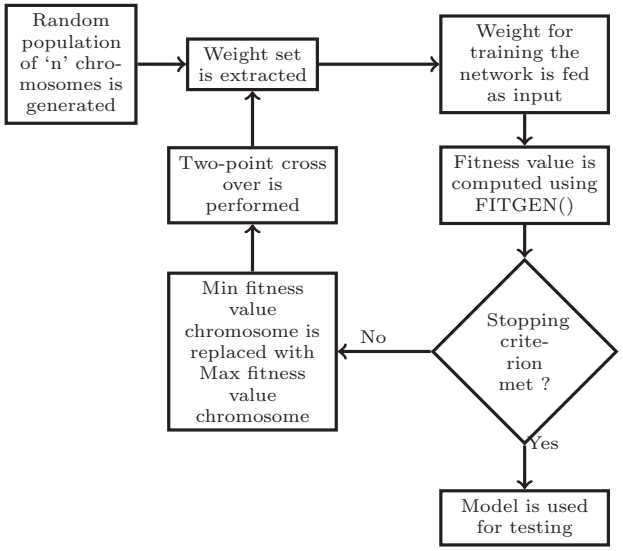


**Figure 2: Flow chart representing Neuro-GA execution**

# 5. PARALLEL COMPUTING OF NEURAL NETWORK

A number of parallel computing algorithms are used to accelerate the training procedure of the neural network model [10]. These approaches can be categorized into node parallelism and training dataset parallelism.

## 5.1 Node parallelism

Node parallelism is based on the structure of neural network and it can be achieved by mapping neurons into different computing nodes for pipelining the process. Each computing node takes charge of only a part of the computation of the neural network.

## 5.2 Training dataset parallelism

In training dataset parallelism, each computing node has the complete neural network in local, and conducts computation for entire neural network. The training dataset is divided into several subsets, and these subsets are assigned to different computing nodes for parallel processing.

In this paper, the concept of training dataset parallelism is used for parallel computing for training the neural network. Figure 3 shows the block diagram for parallel computing of the neural network, comprising of four computing nodes. Each computing node performs full training of the network for one set of the training dataset. This concept is based on master-slave approach. Master uses 'scheduler' for distributing the job among the available computing nodes.

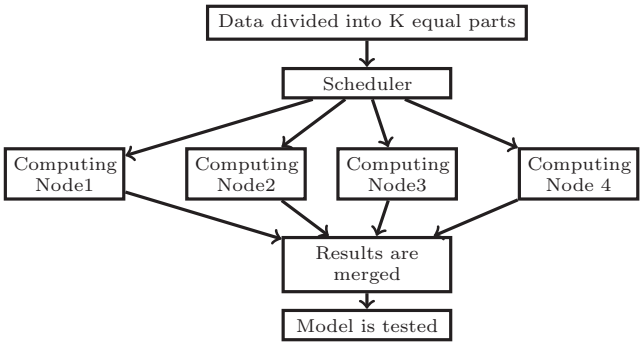


**Figure 3: Flow chart representing Training set parallelism**

# 6. PERFORMANCE EVALUATION PARAMETERS

Software maintainability estimation accuracy for a model designed by using AI techniques is determined by using performance evaluation parameters such as: Mean Relative Error (MRE), Mean Absolute Relative Error (MARE), and Standard Error of the Mean (SEM) [23]. Parameters like True error (e) and Estimate of true error ($\hat{e}$) are being used for evaluating models involving cross validation approach [16].

- **Mean Absolute Error (MAE)**

$$MAE = \frac{1}{n}\sum_{i=1}^{n}(|y_i' - y_i|) \tag{14}$$

- **Mean Absolute Relative Error (MARE)**

$$MARE = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_i - y_i'|}{y_i} \quad (15)$$

In equation 15, a numerical value of 0.05 is added in the denominator in order to avoid numerical overflow (division by zero). The modified MARE is formulated as:

$$MARE = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_i - y_i'|}{y_i + 0.05} \quad (16)$$

- **Standard Error of the Mean (SEM)**

$$SEM = \frac{SD}{\sqrt{n}} \quad (17)$$

where SD is the sample standard deviation, and n is the number of samples.

- **True Error (e)**

$$E_i = \frac{1}{L} * \sum_{j=1}^{L} |y_i(j) - y_i'(j)| \quad (18)$$

where K, L represent the number of folds and number of samples in each fold respectively.

- **Estimate of True Error ($\hat{e}$)** Estimate of the true error which is equal to average of the performances of the k models and is given as:

$$\hat{e} = \frac{1}{K} * \sum_{j=1}^{K}(\hat{e_j}) = \frac{1}{K} * \sum_{j=1}^{K}(\frac{1}{n_j} * \sum_{i=1}^{n_j}(y_i - y_i')^2) \quad (19)$$

where $n_j$ is number of sample in $D_j$ dataset.

# 7. EXPERIMENT AND RESULT

Software maintainability estimation accuracy for the model designed by using AI techniques is determined by using performance evaluation parameters such as Mean Relative Error (MRE), Mean Absolute Relative Error (MARE), and Standard Error of the Mean (SEM) [23]. Parameters like True error (e) and Estimate of true error ($\hat{e}$) are being used for evaluating models involving cross validation approach [16]. Further, this paper employs feature reduction techniques such as PCA and RSA to minimize the features and study the effect on these techniques on the given case studies.

## 7.1 Feature extraction using PCA

In the analysis, the extracted set of the Object-Oriented metrics suite attributes obtained by applying principle component analysis (PCA) are used as input for designing the model to estimate the maintainability of software system. The principal component extraction analysis and varimax rotation concept is applied on all Object-Oriented metrics. Rotated component metrics are tabulated in Table 6. Table 6 shows the relation between the original Object-Oriented metrics and the domain metrics. The value greater then 0.7 (shown bold in Table 6) are the metrics, which are used to interpret the principal component. Table 6 also shows the eigenvalue, variance percentage and cumulative percentage.

**Table 6: Rotated principle component**

| | UIMS | | QUES | | |
|---|---|---|---|---|---|
| metrics | PC1 | PC2 | PC1 | PC3 | PC3 |
| WMC | **0.919** | 0.189 | **0.830** | 0.261 | -0.272 |
| DIT | -0.368 | **0.745** | 0.062 | 0.027 | **0.976** |
| NOC | 0.349 | **-0.792** | - | - | - |
| RFC | **0.957** | 0.182 | **0.874** | 0.339 | 0.048 |
| LCOM | **0.819** | 0.220 | **0.870** | -0.153 | 0.058 |
| MPC | **0.693** | 0.389 | -0.026 | **0.966** | 0.037 |
| NOM | **0.936** | 0.020 | **0.972** | -0.129 | 0.094 |
| DAC | **0.707** | -0.311 | **0.797** | 0.028 | 0.425 |
| SIZE1 | **0.933** | 0.234 | **0.973** | -0.090 | 0.187 |
| SIZE2 | **0.788** | -0.357 | **0.808** | 0.481 | -0.090 |
| Eigenvalues | 5.332 | 2.351 | 5.375 | 1.397 | 1.246 |
| % variance | 53.315 | 23.57 | 59.725 | 15.519 | 13.842 |
| Cumulative % variance | 53.315 | 76.887 | 59.725 | 75.244 | 89.086 |

## 7.2 Feature reduction using RSA

In the analysis of predicting maintainability, the reduced attribute set of the Object-Oriented metrics suite is used. This reduced attribute set was obtained by applying rough set analysis (RSA). But in order to obtain this set of reduced features, data needs to be classified. K-means clustering was used to classify the data in this thesis work. Here, the data available in particular cluster are grouped under a single class. So, after applying K-means clustering, three clusters were obtained and the data were categorized into three groups such as Low, Medium and High. Table 7 shows the cluster center's for the respective metrics of UIMS and QUES software system.

**Table 7: Cluster Center of software metrics using K-Means Clustering**

| | UIMS | | | QUES | | |
|---|---|---|---|---|---|---|
| | C1 | C2 | C3 | C1 | C2 | C3 |
| WMC | 1.60 | 11.23 | 44.33 | 6.01 | 32.35 | 68.33 |
| DIT | 0.75 | 1 | 3.06 | 0.90 | 2 | 3.25 |
| NOC | 0 | 1.80 | 5.60 | - | - | - |
| RFC | 11.80 | 33.10 | 69.75 | 35.47 | 70.05 | 129.75 |
| LCOM | 3.75 | 7.75 | 25.66 | 4.40 | 14.16 | 24.87 |
| MPC | 1.13 | 4.56 | 9.87 | 8.90 | 18.68 | 34.11 |
| DAC | 0.45 | 3.33 | 17.00 | 1.79 | 7.62 | 25.00 |
| NOM | 5.047 | 12.61 | 34.80 | 5.55 | 21.29 | 37.77 |
| SIZE1 | 30.88 | 100.266 | 348.50 | 176.32 | 309.29 | 611.45 |
| SIZE2 | 6.44 | 18.55 | 43.40 | 8.73 | 28.80 | 52.14 |
| CHANGE | 21.14 | 188.33 | 271 | 33.95 | 84.15 | 173.80 |

After categorizing the data into three groups, now the range of each group is determined. Here,

i. Low: Data lies in the range from zero to the average of clusters c1 and c2.

ii. Medium: Data lies in the range from average of clusters c1, c2 and the average of c2 and c3.

iii. High: Data lies in the range from average of the clusters c2, c3 and the maximum values of the respective attributes.

Table 8 and Table 9 contain the tabulated range values of the group of the software metrics.

Rough set obtained a reduced attribute set for estimating maintainability of UIMS ad QUES software system. The reduced attribute set of metrics obtained after applying RSA for UIMS and QUES are tabulated in Table 10.

Table 8: set value UIMS

| | LOW | MEDIUM | HIGH |
|---|---|---|---|
| WMC | [0 6.41) | [6.41 29.78) | [27.78 79.00] |
| DIT | [0 1.37) | [1.37 2.53) | [2.53 4.00] |
| NOC | [0 0.90) | [0.90 3.70) | [3.70 8.00] |
| RFC | [0 21.45) | [21.45 51.42) | [51.42 101.00] |
| LCOM | [0 5.75) | [5.75 16.70) | [16.70 31.00] |
| MPC | [0 2.84) | [2.84 7.21) | [7.21 12.00] |
| NOM | [0 8.83) | [8.83 23.70) | [23.70 40.00] |
| DAC | [0 1.89) | [1.89 10.16) | [10.16 21.00] |
| SIZE1 | [0 65.57) | [65.57 224.38) | [224.38 439.00] |
| SIZE2 | [0 12.49) | [12.49 30.97) | [30.97 61.00] |
| CHANGE | [0 104.73) | [104.73 229.66) | [229.66 289.00] |

Table 9: set value QUES

| | LOW | MEDIUM | HIGH |
|---|---|---|---|
| WMC | [0 19.18) | [19.18 50.34) | [50.34 83.00] |
| DIT | [0 1.45) | [1.45 2.62) | [2.62 4.00] |
| NOC | - | - | - |
| RFC | [0 52.76) | [52.76 99.90) | [99.90 156.00] |
| LCOM | [0 9.28) | [9.28 19.52) | [19.52 33.00] |
| MPC | [0 13.79) | [13.79 26.39) | [26.39 42.00] |
| NOM | [0 13.42) | [13.42 29.53) | [29.53 57.00] |
| DAC | [0 4.71) | [4.71 16.31) | [16.31 25.00] |
| SIZE1 | [0 242.80) | [242.80 460.20) | [460.20 1009.00] |
| SIZE2 | [0 18.61) | [18.61 40.32) | [40.32 82.00] |
| CHANGE | [0 59.05) | [59.05 128.97) | [128.97 217.00] |

Table 10: Reduciced Attribute

| Project | Metrics |
|---|---|
| UIMS | MPC, DAC, SIZE2 |
| QUES | LCOM, DAC, SIZE1 |

## 7.3 Parallel computing Concepts

To carry out this work of parallel computation, hardware requirements such as a Core i5 processor having 2GB RAM, a storage memory of 250GB. Here computing node is process that runs on a physical core.

In this paper, based on the master-slave concept, varying number of computing nodes are considered ranging from one to five. The 5-fold cross validation was performed on both the data sets viz., QUES and UIMS.

- Each fold of the 5 fold data set is assigned to each and every computing node when there are five computing nodes.
- In case, there exists four computing nodes, then the master assigns two folds of the data set to the first computing node, and then assigns one fold of data set to the remaining computing nodes.
- When there are three computing nodes, the master assigns two folds of data sets to the first two computing nodes, and then assigns the last fold to the remaining computing node.
- Similarly, when there are two computing nodes, then the master assigns the first three folds of data set to the first computing node, and the reaming two folds to the second computing node.
- Finally, if there exists a single computing node, then the master assigns all the 5 fold data set to the existing single computing node.

Figure 4 shows the variation of training period based on the number of computing nodes assigned by the master. From Figure 4, it can be inferred that when a single node is used the execution time is much more when compared to the normal serial execution process. Further, it is observed the training time is reduced when the number of computing nodes are increased.

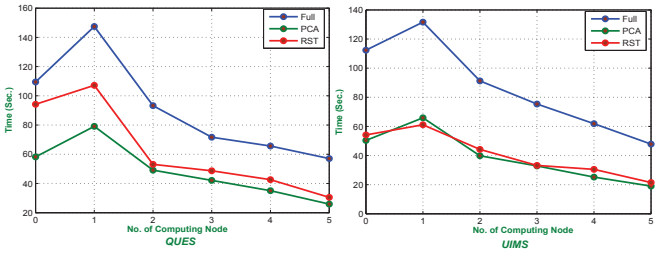


**Figure 4: Training time vs No. of Computing Node**

## 7.4 Neuro-GA

Following are the numerical values used in execution of Neuro-GA approach for predicting software maintainability.

i. Initialization of chromosome: Let the population of size N=50 is considered, initially generated by random process.

ii. Extraction of weight: Each chromosome contains the weight of input to hidden node and hidden node to output. Weight is extracted using Equation 10.

iii. Computing fitness value: The fitness of individual chromosomes is found using the proposed algorithm *FITGEN*. This algorithm is executed with an aim of minimizing the mean square error.

iv. Ranking of chromosomes: The chromosomes in the pool are ranked based on their fitness value. Minimum fitness value chromosome is stripped of by Maximum fitness value chromosome.

v. Crossover: Two point cross-over approach is performed.

vi. Stopping criteria: The execution of the proposed algorithm terminates when 95% of the chromosomes in the pool obtain unique fitness value, beyond this level the fitness value of chromosome get almost saturated.

In this paper, 5-fold cross-validation concept has been considered for both QUES and UIMS for comparing the models. Table 11 and Table 12, show the obtained performance metrics for UIMS and QUES software products.

From Table 11 and Table 12, it can be concluded that the performance in estimating software maintainability is better when RSA in UIMS (A3 analysis) and PCA in QUES (A2 analysis) is considered. The high value of 'R' shows the strong affinity between software metrics and maintainability. Figure 7.4 shows the variance of number of chromosomes

**Table 11: Performance matrix for UIMS data set**

| | r | Epochs | MAE | MARE | SEM |
|---|---|---|---|---|---|
| A1 | 0.8831 | 76 | 0.0831 | 0.3155 | 0.0635 |
| A2 | 0.7921 | 87 | 0.118 | 0.4792 | 0.0191 |
| A3 | 0.9631 | 70 | 0.0439 | 0.1883 | 0.0437 |

**Table 12: Performance matrix for QUES data set**

| | r | Epochs | MAE | MARE | SEM |
|---|---|---|---|---|---|
| A1 | 0.8831 | 87 | 0.1296 | 0.3775 | 0.0197 |
| A2 | 0.7529 | 68 | 0.1039 | 0.3536 | 0.0185 |
| A3 | 0.7409 | 71 | 0.1124 | 0.3884 | 0.0177 |

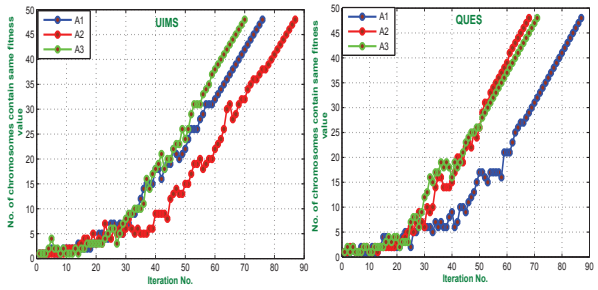


**Figure 5: No. of chromosomes contain same fitness value VS Iteration No.**

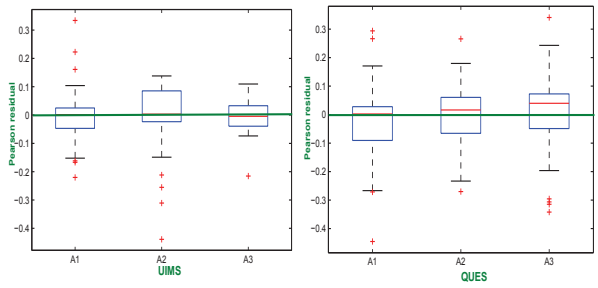


**Figure 6: Residual boxplot for UIMS and QUES**

having same fitness value and iteration number of UIMS and QUES.

Figure 6 shows the Pearson residual boxplots for all three analysis (A1 to A3) for the normalized data of UIMS and QUES data set, allowing a visual comparison. The line in the middle of each box represents the median of the Pearson residual. From Figure 6, it is clear that:

- In case of UIMS, all analysis have a median residual close to zero. Of all the analysis, A3 analysis has the narrowest box and the smallest whiskers, as well as the few number of outliers. Based on these boxplots, it is evident that A3 analysis present best estimation accuracy as compared to other two analysis i.e., A1 and A2. Hence it is observed that the model designed by rough set theory as feature reduction technique yields better maintenance accuracy value.

- In case of QUES, A1 and A2 analysis have a median residual close to zero but in A3 analysis it is observed that median residual is slightly greater then zero. It interprets that the data is slightly underestimated. Of all the analysis, A2 analysis has the narrowest box and the smallest whiskers, as well as the few number of outliers. Based on these boxplots, it is evident that A2 analysis presents best estimation accuracy as compared to other two analysis i.e., A1 and A3. Hence it is observed that the model designed by principal component analysis as feature extraction technique yields better maintenance accuracy value.

## 8. COMPARISON OF MODELS

Apart from the comparative analysis done to find the suitable model which can predict the best software maintainability, this paper also makes the comparison of the proposed work with the work done by Yuming Zhou *et al.* [28] and Van Koten *et al.* [27]. Yuming Zhou *et al.* and Van Koten *et al.* have used same dataset i.e., UIMS and QUES for predicting maintainability based on different regression and neural network models . They have considered 'MMRE' as a performance parameter to compare the models designed for predicting maintainability of Object-Oriented software systems. Table 13 shows the MMRE value of the proposed work and the work done by Yuming Zhou *et al.* and Van Koten *et al.* From Table 13, it can be observed that, in case of QUES software MMRE value is almost same, but in case of UIMS software, the proposed approach obtained better prediction rate for maintainability.

**Table 13: Performance based on MMRE for UIMS and QUES**

| MMRE | | | |
|---|---|---|---|
| Author | Technique | UIMS | QUES |
| Van Koten *et al.*[27] | Bayesian Network | 0.972 | 0.452 |
| | Regression Tree | 1.538 | 0.493 |
| | Backward Elimination | 2.586 | 0.403 |
| | Stepwise Selection | 2.473 | 0.392 |
| Zhou *et al.* [28] | Multivariate linear regression | 2.70 | 0.42 |
| | Artificial neural network | 1.95 | 0.59 |
| | Regression tree | 4.95 | 0.58 |
| | SVR | 1.68 | 0.43 |
| | MARS | 1.86 | 0.32 |
| | A1 | 0.3155 | 0.3775 |
| | A2 | 0.4792 | 0.3536 |
| | A3 | 0.1883 | 0.3884 |

## 9. CONCLUSION

In this paper, an attempt has been made to use software metrics in order to predict software maintainability. Neuro-GA approach coupled with the concept of parallel computing concept was used to design an estimation model by employing 5-fold cross-validation technique for both QUES and UIMS case studies. The concept involved in usage of varying number of computing nodes was explored in this analysis.

Software metrics in combination with feature reduction techniques such as PCA and RSA, were used to analyze the effectiveness of the designed models to estimate the software manintainability for QUES and UIMS software products. These techniques have the ability to predict the output based on the available historical data. Software metrics were taken as input data to train the network, and estimate the

maintainability of the software product. From this analysis, it can be concluded that Neuro-GA approach obtained promising results when compared with the work carried out by Yuming Zhou *et al.* and Van Koten *et al.*. The comparative analysis highlights that there exists a strong relationship between software metrics, and maintainability. It was also observed that, the training time gets reduced to a significant amount when the number of computing nodes were increased.

Further, work can be replicated in estimating the maintainability of the software products by coupling neural network with other techniques such as particle swarm optimization, Fuzzy logic, Clonal selection algorithm etc. Another perspective would be to extend the proposed work to provide 'estimation' as a service using the cloud concept.

# 10. REFERENCES


[1] F. B. E. Abreu and R. Carapuca. Object-Oriented software engineering: Measuring and controlling the development process. In *Proceedings of the 4th International Conference on Software Quality*, volume 186, 1994.

[2] R. K. Bandi, V. K. Vaishnavi, and D. E. Turk. Predicting maintenance performance using object-oriented design complexity metrics. *IEEE Transactions on Software Engineering*, 29(1):77–87, 2003.

[3] A. B. Binkley and S. R. Schach. Validation of the coupling dependency metric as a predictor of run-time failures and maintenance measures. In *Proceedings of the 20th international conference on Software engineering*, pages 452–455. IEEE Computer Society, 1998.

[4] L. C. Briand, J. Wüst, J. W. Daly, and D. V. Porter. Exploring the relationships between design measures and software quality in Object-Oriented systems. *The Journal of Systems and Software*, 51(3):245–273, May 2000.

[5] C. Burgess and M.Lefley. Can genetic programming improve software effort estimation. *Information and Software Technology*, 43:863–873, 2001.

[6] J.-C. Chen and S.-J. Huang. An empirical analysis of the impact of software development problem factors on software maintainability. *Journal of Systems and Software*, 82(6):981–992, 2009.

[7] S. R. Chidamber and C. F. Kemerer. A metrics suite for Object-Oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, June 1994.

[8] D. Coleman, D. Ash, B. Lowther, and P. Oman. Using metrics to evaluate software system maintainability. *IEEE Computer*, 27(8):44–49, 1994.

[9] D. Coleman, B. Lowther, and P. Oman. The application of software maintainability models in industrial software systems. *Journal of Systems and Software*, 29(1):3–16, 1995.

[10] R. Gu, F. Shen, and Y. Huang. A parallel computing platform for training large scale neural networks. In *2013 IEEE International Conference on Big Data*, pages 376–384, 2013.

[11] M. Halstead. *Elements of Software Sciencel*. Elsevier Science, New York, USA, 1977.

[12] B. Henderson-Sellers. *Software Metrics*. Prentice-Hall, UK, 1996.

[13] H.-W. Jung, S.-G. Kim, and C.-S. Chung. Measuring software product quality: A survey of iso/iec 9126. *IEEE software*, 21(5):88–92, 2004.

[14] B. K. Kang and J. M. Bieman. Cohesion and reuse in an Object-Oriented system. In *Proceedings of the ACM SIGSOFT Symposium on software reuseability*, pages 259–262. Seattle, March 1995.

[15] J. Kaur, S. Singh, K. S. Kahlon, and P. Bassi. Neural network-a novel technique for software effort estimation. *International Journal of Computer Theory and Engineering*, 2(1):17–19, 2010.

[16] Kim and Ji-Hyun. Estimating classification error rate: Repeated cross-validation, repeated hold-out and bootstrap. *Computational Statistics and Data Analysis*, 53(11):3735–3745, 2009.

[17] R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence, San Mateo*, pages 1137–1143, 1995.

[18] A. Lake and C. Cook. Use of factor analysis to develop oop software complexity metrics. In *Proceedings of 6th Annual Oregon Workshop on Software Metrics, Silver Falls, Oregon*, 1994.

[19] W. Li and S. Henry. Maintenance metrics for the Object-Oriented paradigm. In *Proceedings of First International Software Metrics Symposium*, pages 52–60, 1993.

[20] M. Lorenz and J. Kidd. *Object-Oriented Software Metrics*. Prentice-Hall, NJ, Englewood, 1994.

[21] T. J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, 2(4):308–320, December 1976.

[22] T. Menzies, B. Caglayan, Z. He, E. Kocaguneli, J. Krall, F. Peters, and B. Turhan. The promise repository of empirical software engineering data, June 2012.

[23] T. Menzies, Z. Chen, J. Hihn, and K. Lum. Selecting best practices for effort estimation. *IEEE Transactions on Software Engineering*, 32(11):883–895, 2006.

[24] P. Oman and J. Hagemeister. Construction and testing of polynomials predicting software maintainability. *Journal of Systems and Software*, 24(3):251–266, 1994.

[25] Z. Pawlak. Rough sets. *International Journal of Computer and Information Sciences*, 11(5):341–356, 1982.

[26] S. L. Schneberger. Distributed computing environments: effects on software maintenance difficulty. *Journal of Systems and Software*, 37(2):101–116, 1997.

[27] C. Van Koten and A. Gray. An application of bayesian network for predicting object-oriented software maintainability. *Information and Software Technology*, 48(1):59–67, 2006.

[28] Y. Zhou and H. Leung. Predicting object-oriented software maintainability using multivariate adaptive regression splines. *Journal of Systems and Software*, 80(8):1349–1361, 2007.