Mr. Sandeep Sharawat / International Journal of Engineering Research and Applications (IJERA)
ISSN: 2248-9622 www.ijera.com
Vol. 2, Issue 2,Mar-Apr 2012, pp.750-755

# Software Maintainability Prediction Using Neural Networks

**Mr. Sandeep Sharawat**
(University School of Information Technology
GGSIPU
Sector 16-C, Dwarka, Delhi)

## Abstract

To increase software maintainability, the main focus on MI (Maintainability Index) which is a composite metric that incorporates a number of traditional source code metrics into a single number that indicates relative maintainability. Li-Henry data is used for the prediction of MI to train neural networks. There are different training algorithms (trainlm, traingdm, trainscg, trainbfg, traincgp, trainoss, trainr, trainrp), even custom training algorithms may be used. After training the neural network, a set of input values are provided & change or say, MI can be determined as output of the trained neural network. Input values may be provided with different changes in input attributes to see, "how MI is affected by different attributes & how should we need to design/code an application to get better maintainability.

**Keywords –** Li-Henry data, Metrics, MI, Neural networks, software maintenance.

## 1. Introduction

Software Maintenance in software engineering is the modification of a software product after delivery to correct faults, to improve performance or other attributes[2,3,5].

The key software maintenance issues are both managerial and technical. Key management issues are: alignment with customer priorities, staffing, which organization does maintenance, estimating costs. Key technical issues are: limited understanding, impact analysis, testing, and maintainability measurement.

Maintainability:- The ease with which a software system or component can be modified or correct faults, improve performance or other factors, or adapt to a changed environment [3,5].

Consistent with these definitions, the maintenance process can be divided into four areas of focus :

a) *Corrective maintenance*: Maintenance performed to correct faults in hardware or software.
b) *Adaptive maintenance*: Software maintenance performed to make a computer program usable in a changed environment.
c) *Perfective maintenance*: Software maintenance performed to improve the performance, maintainability, or other attributes of a computer program.
d) *Preventive maintenance*: Concerns activities aiming on increasing software maintainability and prevent problems in the future.

### 1.1 RELATED WORK

There are different techniques for prediction from existing data in terms of independent variables and dependent variable. In object oriented system, for prediction of maintainability different metrics are used in different researches. Metrics described in research of Shyam R. Chidamber and Chris F. Kemerer are weighted methods per class (WMC), depth of inheritance tree (DIT), number of children (NOC), coupling between object class (CBO), response for a class (RFC), lack of cohesion in methods (LCOM)[8]. Whereas metrics considered in the research work of Dr. Arvinder Kaur, Kamaldeep Kaur and Dr. Ruchika Malhotra are LCOM (lack of cohesion), DIT (depth of inheritance tree), WMC (weighted methods per class), NOC (number of children), RFC (response of class), DAC (data abstraction coupling), MPC (message passing coupling), NOM (number of methods per class)[1]. We considered the metrics used by Li-Henry in their work and feedforward neural networks are used for predictions of MI using the data collected on UIMS (User Interface Management System) and metrics are DIT (Depth in the Inheritance Tree), NOC (Number of Children), MPC (Message Passing Coupling), RFC (Response For Class), LCOM (Lack of Cohesion of Methods), DAC (Data Abstraction Coupling), WMC (Weighted Method Complexity), NOM (Number of Methods), Size1(Number of semicolons per class), Size2 (Number of methods plus number of attributes)[6] as the independent variables & one metric named as Change (Number of lines changed per class in its maintenance history)[6] as dependent variable on the independent variables. This change attribute that indicates relative maintainability which is referred to be MI. In the following subsections we will describe MI, Predictions and Neural Networks and we will use the Neural Networks for prediction of MI using the data collected for UIMS application [6] by W. Li & S. Henry for the above mentioned metrics for training of neural network to get properly weighted & biased trained neural network. In the same way we will design different neural network using different training functions, then we select five test data with target values and record the errors as difference between the output generated and target values. After comparing the results we get the proper trained network using which we can analyze MI to see the effects of change in any metric.

## 1.2 MAINTAINABILITY INDEX

To increase software maintainability, we mainly focus on MI (Maintainability Index) which is a composite metric that incorporates a number of traditional source code metrics into a single number that indicates relative maintainability[3,4].

Software practitioners have been collecting metrics from source code to understand the effects of changes on maintainability of software systems. Maintainability Index (MI) is considered to be a composite metric for the said purpose as a single number that indicates relative maintainability. As originally proposed by Oman and Hagemeister, the MI is comprised of weighted Halstead metrics (effort or volume), McCabe's Cyclomatic Complexity, lines of code (LOC), and number of comments[3]. Two equations were presented: one that considered comments and one that did not. The original polynomial equations defining MI are as follows:

3-Metric:

$$MI=171-3.42\ln(aveE)-0.23aveV(g')-16.2\ln(aveLOC) \quad \ldots\ldots(1)$$

4-Metric:

$$MI=171-3.42\ln(aveE)-0.23aveV(g')-16.2\ln(aveLOC)+0.99aveCM \quad \ldots\ldots(2)$$

where *aveE* is the average Halstead Effort per module, *aveV(g')* is the average extended cyclomatic complexity per module, *aveLOC* is the average lines of code per module, and *aveCM* is the average number of lines of comments per module.

The rationale behind this selection of metrics was to construct a rough order, composite metric that incorporated quantifiable measurements for the following [3]:

• Density of operators and operands (how many variables and how they are used).
• Logic complexity (how many execution paths are in the code).
• Size (how much code is there).
• Human insight (comments in the code).

Other variants of the MI have been evolved using slightly different metrics, metric combinations, and weights. Each has the general flavor of the basic MI equation and underlying rationale.

Reasonable success has been achieved in using MI to quantify and improve software maintainability both during development and maintenance activities [3].

## 1.3 PREDICTIONS

Prediction, commonly known as estimation is an important part of project planning. When estimations are made for projects, these are called effort estimates and when made for the process is called effort estimation or software cost estimation when estimates are made to a maintenances process, the mean of obtaining such estimates is called maintenance cost prediction or maintenance project effort estimation[4].

We used Neural Networks in Matlab software for predictions of maintainability index and predict MI by simulating the neural network.

## 1.4 NEED FOR PREDICTIONS

Software maintenance phase of software development life cycle is an important part or role for providing the software quality attributes such as accuracy and clarity of documentation, modularity, readability, simplicity. It has been observed from many researches in past that software maintainability cost is more than 40 percent of total cost of developing it. The software maintenance of a software system can significantly impact software costs. This means that it is important to be able to forecast a software system's maintainability so to effectively manage costs[4].

Software evolution is not separable from software maintainability which comes into consideration after software is delivered to the client and is in operation mode and changes need to be implemented. Due to the unpredictability of changes needed with time that may be because of varieties of faults, the scope and cost of software maintainability are indefinite after a software product being delivered.

Prediction of MI is considered in the industries for developing software application keeping in the view of software maintenance in such a way that the developed application will be more maintainable.

Also risk factor of cost may be analyzed as if the maintenance phase is more difficult in the SDLC, then it may indicate that it ( i.e. software under prediction) is more difficult to maintain which indirectly means cost of maintaining the software is more. Cost factor may be analyzed in Risk Analysis. After finding that the overall cost of software under development due to high maintainability cost, in the analysis process, it is to be understood that the current design is not much reliable to develop further for current environment. So the design team may be asked to design more reliable design under current circumstances. In this ways, Industries will be benefited by the Prediction of MI in the early phases of development.

## 2. DATA Source

Different researchers attempted to link software matrices to software maintainability in procedural paradigm. Rombach indicates that software maintainability can be predicted by using software matrices. Different object oriented matrices are considered by different researchers in past, we considered, Wei Li and Henry, which also show that software maintainability can be predicted using software matrices. All these preliminary results about the relationship of software matrices & software maintainability were obtained from procedural paradigm. These same metrics have yet to be verified in object oriented paradigm[6].

Study of Li-Henry attempts to bring research in software matrices and the research of object oriented programming together. Specifically, investigates proposed object oriented software matrices and proposes some additional object oriented software matrices & validates the matrices using the maintenance data collected from two commercial software system[6].

The object oriented metrics used by Li-Henry in their research paper are abbreviated as follows[6]:

DIT = Depth in the Inheritance Tree

The DIT metric measures the position of a class in the inheritance hierarchy. One may hypothesize that the larger the DIT metric, the harder it is to maintain the class. The calculation of the DIT metric is the level number for a class in the inheritance hierarchy. The root class DIT is zero, DIT ranges from 0 to N; where N is a positive integer.

NOC = Number of Children

The NOC metric measures the number of direct children a class has. One may intuit that the larger the NOC metric, the harder it is to maintain the class. The calculation of NOC is number of direct sub-classes; ranging from 0 to N; where N is a positive integer.

MPC = Message Passing Coupling

MPC is used to measure the complexity of message passing among classes in the research. MPC is number of send-statements defined in a class. The number of messages sent out from a class may indicate how dependent the implementation of the local methods is upon the methods in other classes.

RFC = Response For Class

The RFC metric measures the cardinality of the response set of a class. One may intuit that the larger the RFC metric, the harder it is to maintain the class since calling a large number of methods in response to a message makes tracing an error difficult. The calculation of RFC is number of local methods and number of methods called by local methods; ranging from 0 to N; where N is a positive integer.

LCOM = Lack of Cohesion of Methods

The LCOM metric measures the lack of cohesion of a class. One may intuit that the larger the metric, the harder it is to maintain the class. The calculation of LCOM is number of disjoint sets of local methods; no two sets intersect; any two methods in the same set share at least one local instance variable; ranging from 0 to N; where N is a positive integer.

DAC = Data Abstraction Coupling

A class can be viewed as an implementation of an ADT(Abstract Data Type). The metric which measures the coupling complexity caused by ADTs is DAC (Data Abstraction Coupling) and is the number of ADTs defined in a class.

WMC = Weighted Method Complexity

WMC metric measures the static complexity of all the methods. The more control flows a class' methods have, the harder it is to understand them, thus the harder it is to maintain them. The WMC is calculated as the sum of McCabe's cyclomatic complexity of each local method; ranging from 0 to N; where N is a positive integer.

NOM = Number of Methods

NOM in a class, since the local methods in a class constitute the interface increment of the class, NOM serves the best as an interface metric. NOM is the number of local methods. The more methods a class has, the more complex the class' interface has incremented.

SIZE1 = number of semicolons per class

It is the traditional Lines of Code metric which is calculated by counting the number of semicolons in a class. The LOC metric is hereby referred to as SIZE1.

SIZE2 = number of methods plus number of attributes

The second size metric used is the number of properties (including the attributes and methods) defined in a class. This size metric is referred to as SIZE2.

The maintenance effort used in the study is (collected for each class maintained):

Change = number of lines changed per class in its maintenance history

The maintenance effort "change" is measured as "the number of lines changed per class. The "change" is used as a dependent variable in this study. A line change could be an addition or a deletion. A change of the content of a line is counted as a deletion and an addition[6].

## 3. Neural Networks

Neural Network Toolbox in MATLAB software provides tools for designing, implementing, visualizing, and simulating neural networks. Neural networks are used for applications where formal analysis would be difficult or

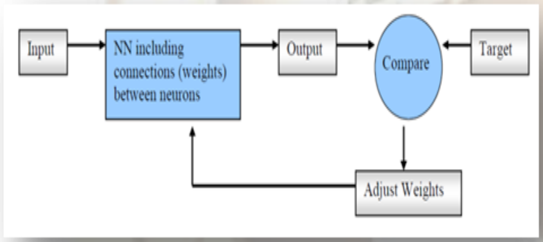
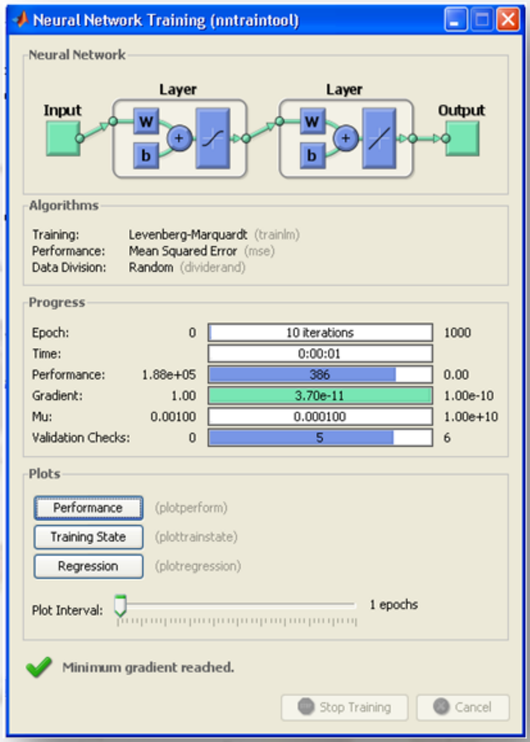


**Fig: a) Working State Diagram of Neural Networks**
**Ref: [**Robyn Ball and Philippe Tissot, "*Demonstration of Artificial Neural Network in Matlab*", Division of Nearhsore Research, Texas A&M University – Corpus Christi]

impossible, such as pattern recognition and nonlinear system identification and control. The toolbox supports feedforward networks, radial basis networks, dynamic networks, self-organizing maps, and other proven network paradigms. We are using feedforward networks out of these, for our work [10].

### 3.1 Machine Training and Learning Functions

Training and learning functions are mathematical procedures used to automatically adjust the network's weights and biases. The training function dictates a global algorithm that affects all the weights and biases of a given network. The learning function can be applied to individual weights and biases within a network[10].



Algorithms

Training: Levenberg-Marquardt (trainlm)
Performance: Mean Squared Error (mse)
Data Division: Random (dividerand)

Progress

| | | | |
|---|---|---|---|
| Epoch: | 0 | 10 iterations | 1000 |
| Time: | | 0:00:01 | |
| Performance: | 1.88e+05 | 386 | 0.00 |
| Gradient: | 1.00 | 3.70e-11 | 1.00e-10 |
| Mu: | 0.00100 | 0.000100 | 1.00e+10 |
| Validation Checks: | 0 | 5 | 6 |

Plots

Performance (plotperform)
Training State (plottrainstate)
Regression (plotregression)

Plot Interval: 1 epochs

✓ Minimum gradient reached.

**Fig: b) Neural Network Training**
[**Snapshot**: Generated during the Training of Neural Network using UIMS application data for training.]

Neural Network Toolbox supports a variety of training algorithms, including several gradient descent methods, conjugate gradient methods, the Levenberg-Marquardt algorithm (LM), and the resilient backpropagation algorithm. The toolbox's modular framework lets us quickly develop custom training algorithms that can be integrated with built-in algorithms. While training our neural network, we can use error weights to define the relative importance of desired outputs, which can be prioritized in terms of sample, timestep (for time-series problems), output element, or any combination of these. One can access training algorithms from the command line or via a graphical tool that shows a diagram of the network being trained and provides network performance plots and status information to help us monitor the training process.

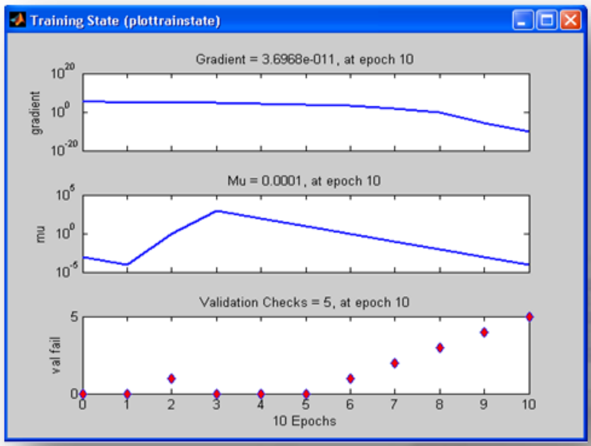


**Fig: c) Training State Graph**
[**Snapshot**: Representing the Training State Graph of the Neural Network.]

A suite of learning functions, including gradient descent, Hebbian learning, LVQ, Widrow-Hoff, and Kohonen, is available in the system and for feedforward networks gradient descent learning functions namely learngd, learngdm.

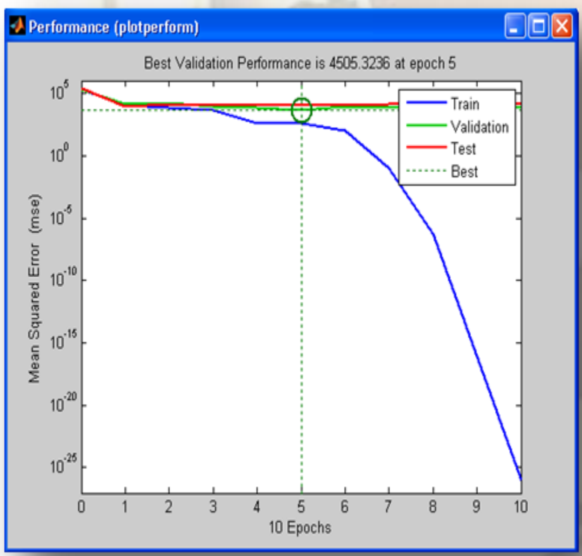


**Fig: d) Performance Graph**
[**Snapshot**: Generated to represent the Performance Graph for the training of Neural Network.]

Whereas a subset of training functions are used, we used training function namely trainlm, traingdm, trainscg, trainbfg, traincgp, trainoss, trainr & trainrp and other training algorithms available are traincgb, trainbr, traincgf, traingd, traingda, traingdx[9,10].

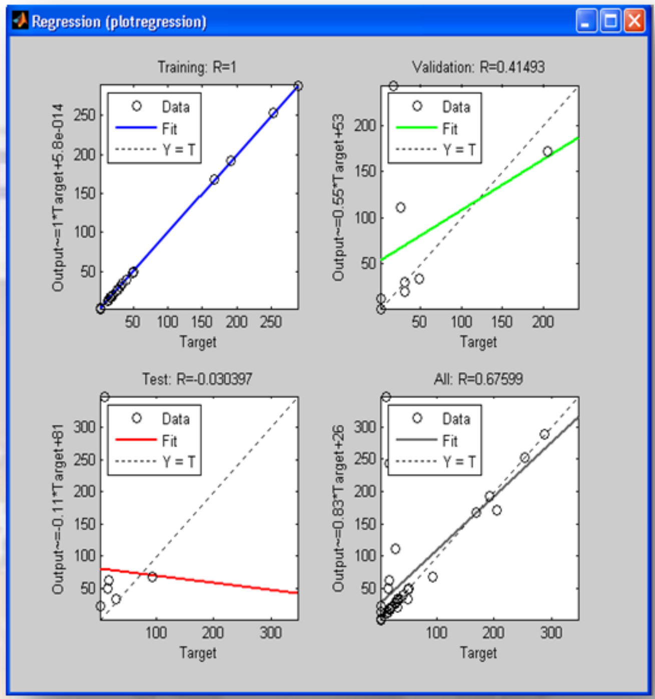


**Fig: d) Regression Graph**
[**Snapshot**: Generated to represent the Regression Graphs for Neural Network]

For our prediction work, we used neural networks in Matlab software system. Neural networks are composed of simple elements operating in parallel. Commonly neural networks are adjusted, or trained, so that a particular input leads to a specific target output. The network is adjusted, based on a comparison of the output and the target, until the network output matches the target. Typically many such input/target pairs are needed to train a network[7,9].

## 3.2 SIMULATION OF NETWORKS

Now we create different neural networks and train them using different training functions using data collect for UIMS application by W. Li and S. Henry[6] to get properly weighted & biased neural networks. Now using these trained network we supply five test data set with known target value and record the difference between the actual output target value as error for each test data set for each neural network which are trained with different training algorithms in Table:1.

| Training Algorithms | Test1 | Test2 | Test3 | Test4 | Test5 |
|---|---|---|---|---|---|
| **Trainlm** | -2.13 | 35.56 | 10.31 | 42.59 | 4.92 |
| **Traingdm** | 203 | 48 | 91 | -36 | 37 |
| **Trainscg** | 89.06 | 3.47 | 2.02 | 185.92 | -10.65 |
| **Trainbfg** | 203 | 48 | 91 | 251 | 37 |
| **Traincgp** | 64.07 | 39.07 | 57.71 | 134.48 | 23.89 |
| **Trainoss** | -83.98 | 48 | 91 | 6.95 | 37 |
| **Trainr** | -84 | -239 | -195.99 | -36 | -250 |
| **Trainrp** | -29.10 | 41.68 | 75.59 | -20.01 | 30.95 |

**TABLE 1**: Resulting error for different neural network.

## 4. CONCLUSION

We observed from Table 1 that Trainlm training function is more suitable than other training functions used. So by training our neural network using Trainlm function, we simulate neural network for the prediction of MI. Now using this trained neural network we can simulate the network for prediction by changing any metric value we can analyze the effect of change in that metric on MI. or MI value can be predicted by increasing or decreasing different metric values like size, depth of inheritance etc.

This work can be used to predict MI in the industries for developing software application keeping in the view of software maintenance in such a way that the developed application will be more maintainable and also can be used in risk or cost analysis as mentioned in Need for Prediction .

In our work, we used neural networks (based on neural networks algorithms), further we can design our own training algorithm for better results focusing on MI, those may be based on Fuzzy Logic algorithms, Artificial Intelligence based algorithms or Support vector based algorithms etc. While designing our own algorithms focusing on MI will result in more accuracy in the prediction of Maintainability Index to show how maintainable software system will be, which may have major impact in industries as companies may be able to predict MI, so that risk analysis or cost analysis can be considered earlier in respect of software maintenance, which will help us to reduce overall cost of software systems.

## References

[1] Dr. Arvinder Kaur, Kamaldeep Kaur, Dr. Ruchika Malhotra, "*Soft computing approaches for Prediction of software Maintenance Effort*", 2010 International Journal of computer application (0975-8887).

[2] Don Coleman and Dan Ash, Bruce Lowther, Paul Oman " *Using Metrices to evaluate software system Maintainability*", Hewlett-Packard, Micron Semiconductor, University of Idaho, IEEE computer, vol.27, no.8, pp.44-49, Aug. 1994.

[3] Kurt D. Welker *"The Software Maintainability Index Revisited*", Idaho National Engineering and Environmental Laboratory, Crosstalk, Aug. 2001.

[4] Riaz, M.; Mendes, E.; Tempero, "*A systematic review of software maintainability prediction and metrics*," Empirical Software Engineering and Measurement, 2009. ESEM 2009. 3rd International Symposium on , vol., no., pp.367-377, 15-16 Oct. 2009

[5] Rikard Land, "*Measurement of Software Maintainability*, Department of computer science, Malardalen University, ARTES Graduate Student Conference, March 2002.

[6] Wei Li and Sallie Henry, "*Object oriented metrics which predict Maintainablity*", Department of computer science, Virginia Polytechnic Institute and State University, Journal of Systems and Software, Volume 23, Issue 2, November 1993, Pages 111-122.

[7] Robyn Ball and Philippe Tissot, " *Demonstration of Artificial Neural Networks in Matlab*", Division of Nearhsore Research, Taxes A&M University, available at URL: http://aiworkshop.tamucc.edu/index_files/NNET%20Demo.pdf .

[8] Shyam R. Chidamber and Chris F. Kemerer, "*A Metrics Suite for Object oriented Design*", IEEE Transactions on Software Engineering, Vol. 20, No. 6, june 1994.

[9] Matlab software information, available at URL: http://www.mathworks.in

[10] Neural network specification, available at URL: http://www.mathworks.in/products/neural-network/description3.html

[11] Li-Henry data for UIMS application with details, available at URL: http://eprints.cs.vt.edu/archive/00000347/01/TR-93-05.pdf