Software Defect Prediction Tool based on Neural Network

Malkit Singh
Student, Department of CSE
Lovely Professional University
Phagwara, Punjab (India) – 144411

Dalwinder Singh Salaria
Assistant Professor, Department of CSE
Lovely Professional University
Phagwara, Punjab (India) – 144411

ABSTRACT

There has been a tremendous growth in the demand for software fault prediction during recent years. In this paper, Levenberg-Marquardt (LM) algorithm based neural network tool is used for the prediction of software defects at an early stage of the software development life cycle. It helps to minimize the cost of testing which minimizes the cost of the project. The methods, metrics and datasets are used to find the fault proneness of the software. The study used data collected from the PROMISE repository of empirical software engineering data. This dataset uses the CK (Chidamber and Kemerer) OO (object-oriented) metrics. The accuracy of Levenberg-Marquardt (LM) algorithm based neural network are comparing with the polynomial function-based neural network predictors for detection of software defects. Our results indicate that the prediction model has a high accuracy.

General Terms

Software defect prediction

Keywords

Defect prediction, Metrics, Neural network, Dataset, Levenberg-Marquardt (LM) algorithm.

1. INTRODUCTION

Software metrics (file-level, class-level, component-level, method-level, process-level and quantitative values-level metrics) are used to find the Software defects in the Software development life cycle before the testing process. Methods used are statistics, machine learning, machine learning along with statistical methods and statistical models vs. expert estimation [1]. Faults in software systems leads to major problem in software. Many of software systems are delivered to users with excessive faults. To find the fault-prone parts of the system and to target those parts for increased quality control and testing is an effective approach to reduction of faults. There are various attributes like defect density, maintainability, fault proneness, normalized rework, understandability and re-usability which determine the quality of the software.

Neural networks are trained to perform complex functions in various fields, including pattern recognition, identification, classification, vision, speech, and control systems. Neural network is based on a machine learning approach. Neural networks can also be trained to solve problems that are difficult for conventional computers or human beings. Neural networks are also used in various fields like data mining [11], image processing [12], diagnostic systems [13] etc.

Levenberg-Marquardt (LM) algorithm is a network training function. According to Levenberg-Marquardt optimization it updates weight and bias values. It is often the fastest back propagation algorithm. It does require more memory than other algorithms.

To predict the fault prone modules for the next release of software, industry uses previous software metrics and fault data [2]. Software fault prediction approaches are much more cost-effective to detect software faults compared to software reviews [2]. Software industry is highly challenged to deliver high quality and reliable software, as the size and complexity of software systems increases. Prediction model for software defects help to minimize the cost and improve the software quality.

2. REVIEW OF LITERATURE

Researchers use Statistical methods (Logistic regression and linear regression) and Machine learning techniques (Decision trees and neural networks) to predict the fault proneness of the code in the Mozilla open source Software system. In their study, performance of Lines of code (LOC) metric is well and correctness of Lack of Cohesion on Methods (LCOM) metric is good but its completeness value is low. For fine grained analysis multivariate models perform better [3].

Design [4] and code metrics [5] are used to compare the accuracy of fault prediction models that are available before and after the system is implemented. Code metrics and design metrics are available only after the system is implemented and before the coding has started [6]. Models are based on the data from one release of a large telecommunication system developed by Ericsson using linear regression. In their study, prediction made after the system is 34% more accurate than before the system. The variability of metrics available before the implementation is 43% and after the implementation is 58% [6] but the performance of the system is same when metrics are not used.

To predict the Software defects, researchers can also apply a machine learning approach on real-time Software systems. Examples include tele-control/ tele-presence, robotics and mission planning systems [7]. There are number of prediction techniques that are used like (Statistical models such as Stepwise Multi-linear Regression models and multivariate models, and machine learning models, such as Artificial Neural Networks, Instance-based Reasoning, Bayesian-Belief Networks, Decision Trees and Rule Induction) but for all the data sets there is no such a technique that gives a accurate result. In their study, for predicting real-time Software defects, size and complexity metrics are not sufficient [7].

Using Statistical and Machine learning methods, researchers predict the Fault-prone Software modules. To find the impact on Static code metrics on fault proneness they compare the logistic regression with Machine learning methods (Artificial Neural Network, Decision Tree, Support Vector Machine (SVM), cascade correlation network, group method of data handling polynomial method, gene expression programming) [8]. To determine the performance by calculating Area under curve (AUC) from a ROC Curve researchers use Receiver operating characteristic (ROC) curves for the modules predicting as fault-prone or not fault-prone [8].

On two large telecommunication systems, researchers compare the accuracy of Statistical prediction models with the expert estimation. They use two approaches, statistical fault prediction model and human experts to predict the Fault-prone of the code units. In the absence of experts, statistical models can be used and they are cheap to build. On small and large systems they perform well. Expert estimation is limited on large systems [9].

To find the detection of software defects, the design of polynomial function-based neural network predictor is used. Polynomial function-based neural networks (pf-NNs) can be divided into two categories are linear function-based neural network (lf-NN) and quadratic function-based neural network (qf-NN). Pf-NNs learned with the use of the standard cost function and weighted cost function. They compare the results with radial basis function (RBF) NNs. In their study, pf-NNs used with standard cost function provides the high accuracy (96.1%) but low probability of detection (66.7 or 63.3%) and pf-NNs used with weighted cost function provides the lower accuracy 80.3% and 78.8% but higher probability of detection 93.3% and 96.7% for lf-NN and qf-NN, respectively [10].

Neural networks are used in various fields like:

In a wide range of supervised and unsupervised learning problems**,** neural network learning algorithms have been successfully applied. Researchers used two classes of approaches for data mining with neural networks. The first type of approach involves extracting symbolic models from trained neural networks, called rule extraction. The second approach is to directly learn simple easy-to-understand networks. In their study, using rule extraction algorithms, to translating the functions represented by trained neural networks into languages that are easier to understand. These methods promote comprehensive stability. Methods that directly learn simple networks are frequently humanly interpretable. To ensuring that the relationship between each input and each output is a simple one they are limited to a single layer of weighted connections [11].

Image preprocessing is a popular application area. Neural networks are also used in image processing. For image reconstruction, image restoration and image enhancement, several ANNs were developed but these networks were not adaptive. When ANNs may reduce the amount of computation well or when existing algorithms fail then neural solutions are truly interesting [12].

A neural network (NN) is used with case-based reasoning (CBR) in a diagnostic system. The neural network is used to make hypotheses to solve a new problem and to guide the CBR module in the search for a similar previous case that supports one of the hypotheses. For the diagnosis of congenital heart diseases (CHD) the NN-CBR model has been used in the development of a system. In their study, to solve problems that cannot be solved by the neural network with a good level of accuracy, the hybrid system manages them accurately [13].

3. PROPOSED METHODOLOGY

The experiments reported here involve ant-1.7 regarding defect problems coming from the PROMISE repository of empirical software engineering data (http://promisedata.googlecode.com). Our objective is to measure the accuracy of the proposed neural network (NN) classifier and compare it with the accuracy of some other classifier reported in the literature.

For the software defects, the experiments completed in this study are reported using a split of data into 85%-15% for training and testing subsets, namely, 15% of the total set of patterns is selected randomly using random number generator for testing purpose and remaining patterns are used for training purposes.

3.1 Ant-1.7 dataset

Ant-1.7 dataset data comes from the PROMISE repository (http://promisedata.googlecode.com). This dataset is one of the Marian Jureckzo Datasets. This dataset uses the CK OO metrics. Table 1 shows the metrics and attributes used in this dataset.

Table 1. Attribute information of Ant-1.7 dataset

Metric	Attribute	Description
Chidamber and Kemerer [14]	WMC	Weighted methods per class
	DIT	Depth of Inheritance Tree
	NOC	Number of Children
	CBO	Coupling between object classes
	RFC	Response for a Class
	LCOM	Lack of cohesion in methods
Henderson-Sellers [15]	LCOM3	Lack of cohesion in methods
Bainsy and Davis [16]	NPM	Number of Public Methods
	DAM	Data Access Metric

	MOA	Measure of Aggregation
	MFA	Measure of Functional Abstraction
	CAM	Cohesion Among Methods of Class
Tang et al. [17]	IC	Inheritance Coupling
	CBM	Coupling Between Methods
	AMC	Average Method Complexity
Martin [18]	Ca	Afferent couplings
	Ce	Efferent couplings
McCabe [19]	CC	Cyclomatic complexity
	Max(CC)	The greatest value of CC
	Avg(CC)	The arithmetic mean of the CC
	LOC	Lines of code

4. EXPERIMENTAL ANALYSIS

4.1 Development of the tool

Graphic user interface (GUI) is developed using MATLAB R2011a. Parameters that are used in this GUI are taken from McCabe, Chidamber and Kemerer, Henderson-Sellers, Bainsy and Davis, Tang et al., Martin and Loc Metrics.

4.2 Preparing the data

Data for classification problems can very often have textual or non-numeric information. Neural networks however cannot be trained with non-numeric data. Hence there need to translate the textual data into a numeric form. There are several ways to translate textual or symbolic data into numeric data.

Some of the common symbol translation techniques used is unary encoding, binary encoding and numbering classes. It can be use unary encoding to perform symbol translation.

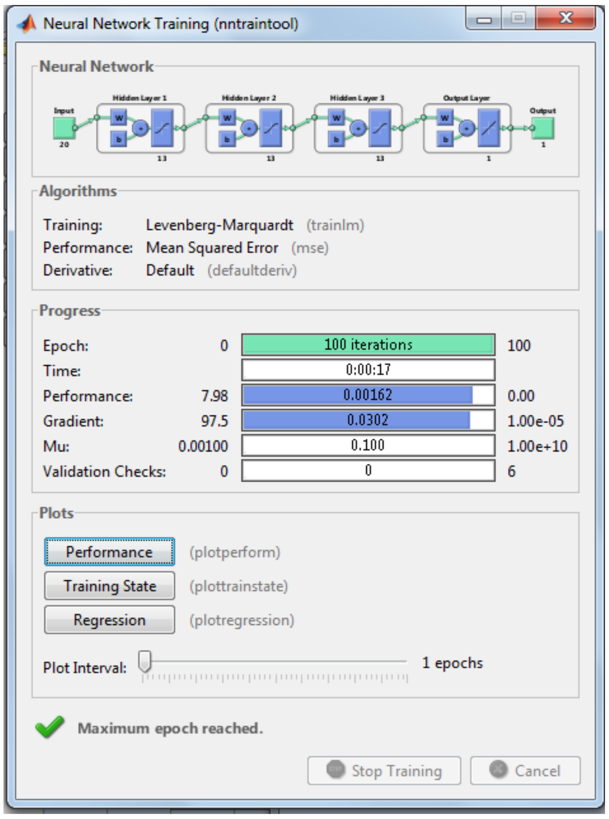
4.3 Building the neural network classifier

The next step is to create a neural network. Since the neural network starts with random initial weights so, the random seed is set to avoid this randomness.

A 3-hidden layers feed forward network is created with 13 neurons in each of the hidden layers. Now the network is ready to be trained.

4.4 Testing the classifier

The trained neural network can now be tested with the testing samples. Before predicting the output, first of all the training of the neural network is required.



Algorithms
Training: Levenberg-Marquardt (trainlm)
Performance: Mean Squared Error (mse)
Derivative: Default (defaultderiv)

Progress

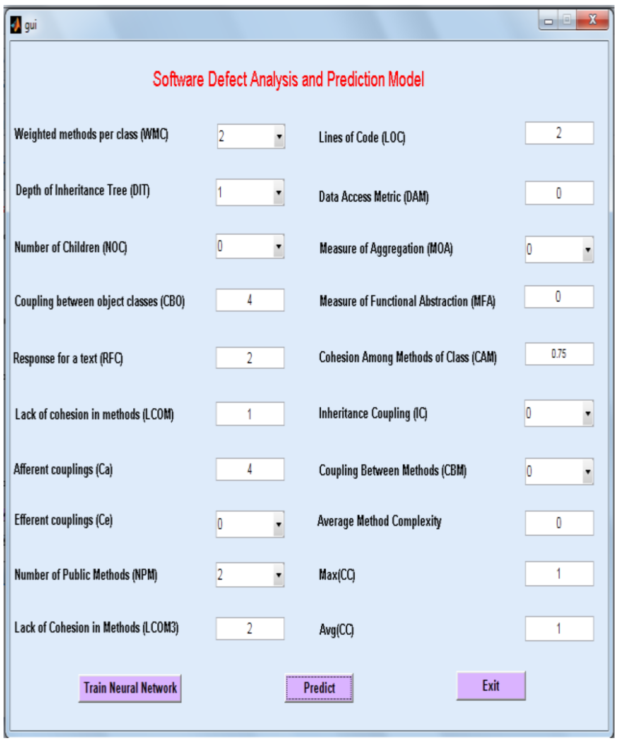
Epoch:	0	100 iterations	100
Time:		0:00:17	
Performance:	7.98	0.00162	0.00
Gradient:	97.5	0.0302	1.00e-05
Mu:	0.00100	0.100	1.00e+10
Validation Checks:	0	0	6

Plots
Performance (plotperform)
Training State (plottrainstate)
Regression (plotregression)
Plot Interval: 1 epochs

Maximum epoch reached.

Fig 1: Training of neural network

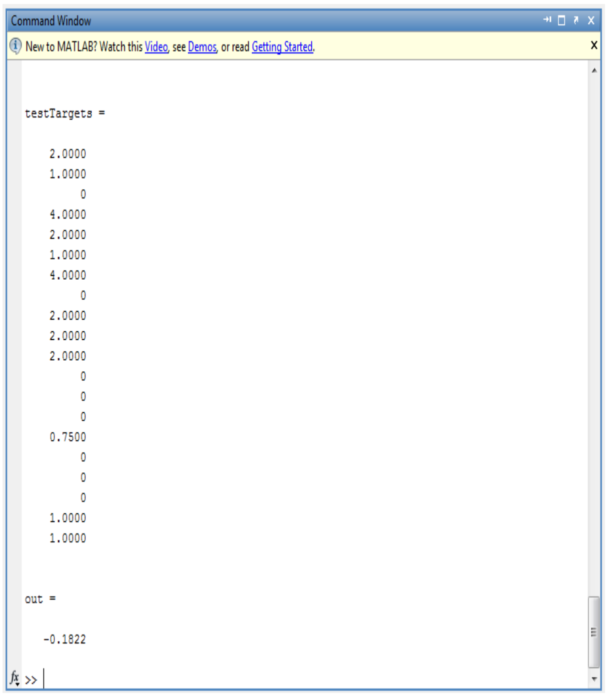
In Fig 1, neural network is trained in which 1 input layer, 3 hidden layers and 1 output layer is used. The algorithm has a specified maximum number of epochs. Upon exceeding this number, the algorithm may be stopped.



Weighted methods per class (WMC)	2	Lines of Code (LOC)	2
Depth of Inheritance Tree (DIT)	1	Data Access Metric (DAM)	0
Number of Children (NOC)	0	Measure of Aggregation (MOA)	0
Coupling between object classes (CBO)	4	Measure of Functional Abstraction (MFA)	0
Response for a text (RFC)	2	Cohesion Among Methods of Class (CAM)	0.75
Lack of cohesion in methods (LCOM)	1	Inheritance Coupling (IC)	0
Afferent couplings (Ca)	4	Coupling Between Methods (CBM)	0
Efferent couplings (Ce)	0	Average Method Complexity	0
Number of Public Methods (NPM)	2	Max(CC)	1
Lack of Cohesion in Methods (LCOM3)	2	Avg(CC)	1

Fig 2: Inputs to the network

In Fig 2, there are 20 inputs and output is calculated by click on the predict button. Inputs are given from the testing data. Train neural network button is used for to train the neural network using training data from the dataset. Exit button is used to exit the Graphical user interface.


Command Window
New to MATLAB? Watch this Video, see Demos, or read Getting Started.

```
testTargets =

    2.0000
    1.0000
         0
    4.0000
    2.0000
    1.0000
    4.0000
         0
    2.0000
    2.0000
    2.0000
         0
         0
         0
    0.7500
         0
         0
         0
    1.0000
    1.0000


out =

  -0.1822

>>
```

Fig 3: Prediction of output

In Fig 3, there is 1 output is predicted using the input values. In the same way, using testing data other outputs are predicted.

5. RESULTS AND COMPARISON

The proposed Software defect prediction model based methodology is implemented in MATLAB R2011a. MATLAB (Matrix Laboratory) environment is one such facility which contributes a high performance language for technical computing. The results are given here.

Table 2. Predicted outputs

serial no.	record no.	Predicted outputs up to two decimal place	actual outputs
1	608	-0.18	0
2	675	-0.19	0
3	96	0.01	0
4	681	0.02	0
5	472	-0.03	0
6	74	-0.02	0
7	209	0.01	0
8	408	-0.02	0
9	64	1.52	6
10	719	0	0
11	119	-0.01	0
12	724	-0.06	0
13	714	0.01	0
14	363	0	0
15	597	1.93	1
16	107	14.77	0
17	315	-1.48	0
18	683	-0.25	1
19	591	0.06	0
20	715	0.01	0
21	489	0.05	1
22	28	-0.41	0
23	633	0.03	0
24	696	0.01	0
25	506	-0.52	0

26	565	-0.03	0
27	554	4.94	0
28	293	3.43	1
29	558	-0.01	0
30	129	0	0
31	527	0.02	0
32	25	0.02	0
33	208	-4.62	3
34	36	-0.24	0
35	395	-0.04	1
36	614	0.01	0
37	518	-6.35	0
38	237	0.06	0
39	708	1.21	0
40	27	-0.02	0
41	328	-0.02	2
42	285	0	0
43	571	2.37	0
44	593	14.82	1
45	141	5.84	1
46	366	-0.01	0
47	333	0.76	4
48	482	-0.02	0
49	529	0.08	0
50	563	-0.02	0
51	207	0.59	0
52	507	-0.01	0
53	331	0.03	0
54	122	0.04	0
55	90	-0.02	0
56	372	-7.22	4
57	716	-0.65	0
58	255	3.92	0
59	437	0.05	1
60	168	10.42	0
61	560	0.79	6
62	191	-0.22	0
63	378	0.02	0
64	522	-0.54	3
65	664	-0.01	1
66	5	-0.01	1
67	409	-0.22	0
68	105	-0.13	0
69	113	-0.03	0
70	193	1.99	3
71	627	-0.03	0
72	110	0.12	0
73	607	-0.01	0
74	183	2.22	1
75	693	2.93	1
76	262	9.5	4
77	148	0.14	0
78	188	0.02	0
79	460	2.77	0
80	354	-0.75	1
81	263	0	0
82	620	0.24	0
83	300	0.01	0
84	410	-0.01	0
85	684	0.01	0
86	214	-0.01	1
87	180	0.07	0
88	562	-0.02	1
89	93	0.03	0
90	424	-7.79	0
91	58	-2.47	3
92	42	0.06	0
93	396	-0.13	0

94	581	-0.03	0
95	312	-0.56	1
96	98	6.37	0
97	425	-0.69	0
98	351	0.02	0
99	10	-0.02	0
100	252	2.21	0
101	Average	0.6043	0.54

Table 2 shows the results of predicted outputs calculated by using testing data from the dataset. Record numbers are calculated by random number generator for testing the data. Actual output is the outputs that are used in the dataset. As a result, the average of predicted outputs and actual outputs is 0.6043 and 0.54. To find out the percentage error and percentage accuracy the formula are given below.

Percentage error = | average predicted outputs – average actual outputs | / average actual outputs *100

Percentage error = 11.91%

Percentage accuracy = 100 – percentage error

Percentage accuracy = 88.09%

In Fig 4, the design of polynomial function-based neural network predictors for detection of software defects [10] claims that the polynomial function-based neural networks (pf-NNs) used with weighted cost function provides the lower accuracy 80.3% and 78.8% for linear function-based neural network (lf-NN) and quadratic function-based neural network (qf-NN) respectively.

But, by using the Levenberg-Marquardt (LM) algorithm based neural network for predicting the software defects provides the better accuracy 88.09%. Our results indicate that the prediction model has a high accuracy.

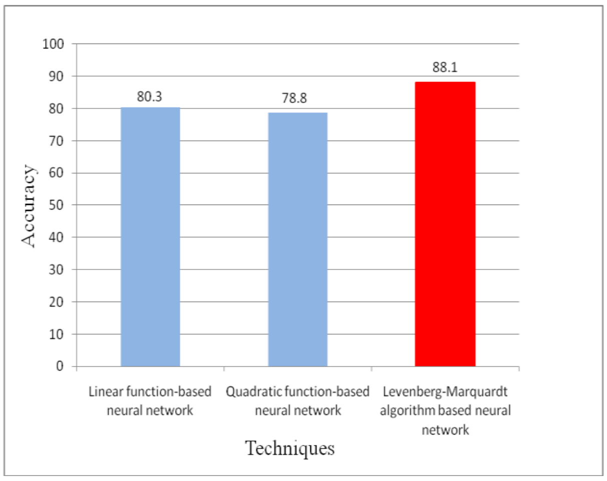


Fig4: Comparison of the accuracy with other models

6. CONCLUSION

It is needed to have a good prediction system for predicting the software defects at an early stage of software development life cycle. Using neural network technique and Levenberg-Marquardt (LM) algorithm, the accuracy of our proposed system is better that polynomial function-based neural network. Neural network is based on a machine learning approach. So, it is found that machine learning models are mostly used and provides the better results.

In future, one can use other training algorithms to increase the accuracy level for predicting the software defects. Increase the use of models which are based on machine learning techniques. Machine learning models have better features than other approaches. Using class level metrics, conduct more studies on fault prediction models. Increase the usage of public datasets for software fault prediction problem.

7. REFERENCE

[1] Catal, C., & Diri, B. (2009). A systematic review of software fault prediction studies. Expert Systems with Applications, 36(4), 7346-7354.

[2] Catal, C. (2011). Software fault prediction: A literature review and current trends. Expert Systems with Applications, 38(4), 4626-4636.

[3] Gyimothy, T., Ferenc, R., & Siket, I. (2005). Empirical validation of object-oriented metrics on open source software for fault prediction. Software Engineering, IEEE Transactions on, 31(10), 897-910.

[4] El Emam, K., Melo, W., & Machado, J. C. (2001). The prediction of faulty classes using object-oriented design metrics. Journal of Systems and Software, 56(1), 63-75.

[5] Zhao, M., Wohlin, C., Ohlsson, N., & Xie, M. (1998). A comparison between software design and code metrics for the prediction of software fault content. Information and Software Technology, 40(14), 801-809.

[6] Tomaszewski, P., Lundberg, L., & Grahn, H. (2005). The accuracy of early fault prediction in modified code. In Proceedings of the Fifth Conference on Software Engineering Research and Practice in Sweden (SERPS) (pp. 57-63).

[7] Challagulla, V. U., Bastani, F. B., Yen, I. L., & Paul, R. A. (2005, February). Empirical assessment of machine learning based Software defect prediction techniques. In Object-Oriented Real-Time Dependable Systems, 2005. WORDS 2005. 10th IEEE International Workshop on (pp. 263-270). IEEE. 4

[8] Singh, Y., Kaur, A., & Malhotra, R. (2010). Prediction of fault-prone software modules using statistical and machine learning methods. International Journal of Computer Applications, 1(22), 8-15.

[9] Tomaszewski, P., Håkansson, J., Grahn, H., & Lundberg, L. (2007). Statistical models vs. expert estimation for fault prediction in modified code–an industrial case study. Journal of Systems and Software, 80(8), 1227-1238.

[10] Park, B. J., Oh, S. K., & Pedrycz, W. (2011). The design of polynomial function-based neural network predictors for detection of software defects. Information Sciences.

[11] Craven, M. W., & Shavlik, J. W. (1997). Using neural networks for data mining. Future generation computer systems, 13(2), 211-229.

[12] Egmont-Petersen, M., de Ridder, D., & Handels, H. (2002). Image processing with neural networks—a review. Pattern recognition, 35(10), 2279-2301.

[13] Reategui, E. B., Campbell, J. A., & Leao, B. F. (1997). Combining a neural network with case-based reasoning in a diagnostic system. Artificial Intelligence in Medicine, 9(1), 5.

[14] Chidamber, S. R., & Kemerer, C. F. (1994). A metrics suite for object oriented design. Software Engineering, IEEE Transactions on, 20(6), 476-493.

[15] Sellers, B. H. (1996). Ojbect-Oriented Metrics. Measures of Complexity.

[16] Bansiya, J., & Davis, C. G. (2002). A hierarchical model for object-oriented design quality assessment. Software Engineering, IEEE Transactions on, 28(1), 4-17.

[17] Tang, M. H., Kao, M. H., & Chen, M. H. (1999). An empirical study on object-oriented metrics. In Software Metrics Symposium, 1999. Proceedings. Sixth International (pp. 242-249). IEEE.

[18] Martin, R. (1994). OO design quality metrics. An analysis of dependencies.

[19] McCabe, T. J. (1976). A complexity measure. Software Engineering, IEEE Transactions on, (4), 308-320.

8. AUTHORS' PROFILE

Malkit Singh received his B.Tech degree in Information Technology from Lovely Professional University, Phagwara, Punjab, INDIA, in 2011, now he is doing M.Tech (CSE) from Lovely Professional University, Phagwara, Punjab, INDIA. His research interests include software engineering.

Dalwinder Singh Salaria received his B.E. degree in CSE from SSJCOE, Jalgaon (MH), INDIA, in 2003 and the M.Tech degree in CSE from Guru Nanak Dev Engineering College, Ludhiana (PB), INDIA, in 2008. He is working as an Assistant Professor with Department of CSE at Lovely Professional University (PB), INDIA. His research interests include databases and Software engineering**.**