World Academy of Science, Engineering and Technology
International Journal of Computer and Systems Engineering
Vol:4, No:7, 2010

Identification of Reusable Software Modules in Function Oriented Software Systems using Neural Network Based Technique

Sonia Manhas, Parvinder S. Sandhu,Vinay Chopra, Nirvair Neeru

***Abstract*—**The cost of developing the software from scratch can be saved by identifying and extracting the reusable components from already developed and existing software systems or legacy systems [6]. But the issue of how to identify reusable components from existing systems has remained relatively unexplored. We have used metric based approach for characterizing a software module. In this present work, the metrics McCabe's Cyclometric Complexity Measure for Complexity measurement, Regularity Metric, Halstead Software Science Indicator for Volume indication, Reuse Frequency metric and Coupling Metric values of the software component are used as input attributes to the different types of Neural Network system and reusability of the software component is calculated. The results are recorded in terms of Accuracy, Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE).

***Keywords*—**Software reusability, Neural Networks, MAE, RMSE, Accuracy.

I. Introduction

SOFTWARE reusability is an attribute that refers to the expected reuse potential of a software component. Software reuse not only improves productivity but also has a positive impact on the quality and maintainability of software products [1]. A component can be considered an independent replaceable part of the application that provides a clear distinct function. A component can be a coherent package of software that can be independently developed and delivered as a unit, and that offers interfaces by which it can be connected unchanged with other components to compose a larger system [1]. According to Gomes [2], the idea of software reuse appeared in 1968, opening new horizons for the software design and development. Reusable software components have been promoted in recent years [3].

There are two approaches for reuse of code: develop the reusable code from scratch or identify and extract the reusable code from already developed code. The organization that has experience in developing software, but not yet used the software reuse concept, there exist extra cost to develop the reusable components from scratch to build and strengthen their reusable software reservoir [4]. The cost of developing the software from scratch can be saved by identifying and extracting the reusable components from already developed and existing software systems or legacy systems [6]. In other words, The software industry is moving toward large-scale reuse, resulting in savings of time and money. To develop a new system from scratch is very costly. This has made custom software development very expensive. It is generally assumed that the reuse of existing software will enhance the reliability of a new software application. This concept is almost universally accepted because of the obvious fact that a product will work properly if it has already worked before. But the issue of how to identify reusable components from existing systems has remained relatively unexplored. In both the cases, whether we are developing software from scratch or reusing code from already developed projects, there is a need of evaluating the quality of the potentially reusable piece of software. The contribution of metrics to the overall objective of the software quality is understood and recognized [7]-[9]. But how these metrics collectively determine reusability of a software component is still at its early stage. A neural Network approach could serve as an economical, automatic tool to generate reusability ranking of software [10]. But, when one designs with Neural Networks alone, the network is a black box that needs to be defined, which is a highly compute-intensive process. One must develop a good sense, after extensive experimentation and practice, of the complexity of the network and the learning algorithm to be used.

In this paper, Neural Network techniques are empirically explored to evaluate the reusability of the function oriented software systems. This paper consists of four sections. The second section explains the steps for identification of reusable software component is discussed. In the third section, implementation results are illustrated and in the final section conclusion is written on the basis of results obtained.

Sonia Manhas is working as HOD IT, SSCET, Badhani, Punjab, India.
Vinay Chopra is working as Lecturer in Deptt. of CSE, DAVIET, Jalandhar, Punjab, India.
Nirvair Neeru is workign as lecturer at UCOE, Punjabi University, Patiala.
Parvinder S. Sandhu is working as Professor with Computer Science & Engineering Department, Rayat & Bahra Institute of Engineering & Bio-Technology, Sahauran, Distt. Mohali (Punjab)-140104 INDIA

II. Proposed Methodology

A. Selection of Metric Suit for Function Oriented Paradigm

A framework of metrics is proposed for structural analysis of procedure or function-oriented. The code of software is parsed to calculate the metric values. The following suits of metrics are able to target those the essential attributes of function oriented features towards measuring the reusability of software modules, so it tried to analyze, refine and use following metrics to explore different structural dimensions of Function oriented components.

The proposed metrics for Function Oriented Paradigm are as follows:

i) Cyclometric Complexity Using Mc Cabe's Measure [11][12]:

According to Mc Cabe, the value of Cyclometric Complexity (*CC*) can be obtained using the following equation:

$$CC = Number\ of\ predicate\ nodes + 1 \quad (1)$$

Where predicate nodes are the nodes of the directed graph, made for the component, where the decisions are made.

Hence, the value of *CC* of a software component should be in between upper and lower bounds as a contribution towards reusability.

If *CC* is high with high regularity of implementation then there exists high functional usefulness.

ii) Halstead Software Science Indicator [11] [13]

According to this metric volume of the source code of the software component is expressed in the following equation:

$$Volume = N1 + N2 \log 2(\eta 1 + \eta 2) \quad (2)$$

Where, $\eta 1$ is the number of distinct operators that appear in the program, $\eta 2$ is number of distinct operands that appear in the program, *N1* is the total number of operator occurrences and *N2* is the total number of operand occurrences.

The high volume means that software component needs more maintenance cost, correctness cost and modification cost. On the other hand, less volume increases the extraction cost, identification cost from the repository and packaging cost of the component. So the volume of the reusable component should be in between the two extremes.

iii) Regularity Metric [11][13]

The notion behind Regularity is to predict length based on some regularity assumptions. As actual length (*N*) is sum of *N1* and *N2*. The estimated length is shown in the following equation:

$$Estimated\ Lenght = N' = \eta 1 \log 2\,\eta 1 + \eta 2 \log 2\,\eta 2 \quad (3)$$

The closeness of the estimate is a measure of the Regularity of Component coding is calculated as:

$$\text{Re}\,gularity = 1 - \{(N - N')/N] = N'/N \quad (4)$$

The above derivation indicates that Regularity is the ratio of estimated length to the actual length. High value of Regularity indicates the high readability, low modification cost and non-redundancy of the component implementation [24].

Hence, there should be some minimum level of Regularity of the component to indicate the reusability of that component.

iv) Reuse-Frequency Metric [11][13]

Reuse frequency is calculated by comparing number of static calls addressed to a component with number of calls addressed to the component whose reusability is to be measured. Let N user defined components be X_1, X_2 … X_N in the system, where S_1, S_2 … S_M are the standard environment components e.g. printf in C language, then Reuse-Frequency is calculated as:

$$Reuse - Frequency = \frac{\eta(C)}{\frac{1}{M}\sum_{i=0}^{M}\eta(S_i)} \quad (5)$$

Equation (5) shows that the Reuse-Frequency is the measure of function usefulness of a component. Hence there should be some minimum value of Reuse- Frequency to make software component really reusable [24].

v) Coupling Metric [11]

Functions/methods that are loosely bound tend to be easier to remove and use in other contexts than those that depend heavily on other functions or non-local data. Different types of coupling effects reusability to different extent.

Data Coupling: Data coupling exists between two functions when functions communicate using elementary data items that are passed as parameters between the two.

Stamp Coupling: When two functions communicate using composite data item e.g. structure in C language then that kind of coupling is called Stamp Coupling.

Control Coupling: If data from one function is said to direct the order of instruction execution in another function then Control Coupling is there between those functions.

Common Coupling: In case of Common Coupling the two functions share global data items. Weight of coupling increases from category "*a*" to "*d*", means

Data Coupling is lightest weight coupling, whereas Content Coupling is the heaviest one.

Let

a_i be the number of functions called and Data Coupled with function "*i*"

b_i be the number of functions called and Stamp Coupled with function "*i*"

c_i be the number of functions called by function "*i*" and Control Coupled with function "*i*"

d_i be the number of functions Common Coupled with function "*i*"

$$f(x,a,c) = \frac{1}{1+e^{-a(w_1 a_i + w_2 b_i + w_3 c_i + w_4 d_i - c)}} \quad (6)$$

Where a = 10, c = 0.5 and w_i for i = 1, 2, 3, 4 is the weights of the respective the coupling types.

As coupling increases, there is decrease in understandability and maintainability, so there should be some maximum value of the coupling.

B. Design & Evaluate Neural Network System

The following five Neural Network algorithms are experimented:

- Batch Gradient Descent
- Batch Gradient Descent with momentum
- Variable Learning Rate
- Variable Learning Rate training with momentum
- Resilient Backpropagation

The following are the steps for each Neural Network based system:

i) Phase I

The following steps will be followed to train a Neural Network:

- Load the data
- Divide data into Training, Validation and Test data
- Set number of hidden neurons
- Training is accomplished by sending a given set of inputs through the network and comparing the results with a set of target outputs.
- If there is a difference between the actual and target outputs, the weights are adjusted to produce a set of outputs closer to the target values.
- Network weights are determined by adding an error correction value to the old weight.
- The amount of correction is determined
- This Training procedure is repeated until the network performance no longer improves.

ii) Phase II

This phase is a Testing phase. In this step the trained Neural Network is evaluated against the testing data on the different criteria as described in the next step.

C. Comparison Criteria

The comparisons are made on the basis of value of *MAE*, *RMSE* and *Accuracy* values of the neural network model. The details of the *MAE* and *RMSE* are given below:

- *Mean absolute error (MAE)*

Mean absolute error, *MAE* is the average of the difference between predicted and actual value in all test cases; it is the average prediction error [14]. The formula for calculating *MAE* is given in equation shown below:

$$MAE = \frac{|a_1 - c_1| + |a_2 - c_2| + \ldots + |a_n - c_n|}{n} \quad (7)$$

Assuming that the actual output is a, expected output is c.

- *Root mean-squared error (RMSE)*

RMSE is frequently used measure of differences between values predicted by a model or estimator and the values actually observed from the thing being modeled or estimated [15]. It is just the square root of the mean square error as shown in equation given below:

$$RMSE = \frac{|a_1 - c_1| + |a_2 - c_2| + \ldots + |a_n - c_n|}{n} \quad (8)$$

D. Conclusions Drawn

The conclusions are made on the basis of the results calculated in the previous section.

III. Implementation And Results

In this paper, the implementation of the algorithm is done in Matlab 7.1 environment and Neural Network toolbox for Matlab is used. The dataset is collected and Batch Gradient Descent, Batch Gradient Descent with momentum, Variable Learning Rate, Variable Learning Rate training with momentum and Resilient Backpropagation based neural networks are experimented to obtain the results in terms of *Accuracy*, *MAE* and *RMSE* values. The same neural network is run for five times and the following table I is showing the Results of five different iterations of different Neural Network Based algorithms for Identification of Reusable Modules in the function based software systems. The table II shows the Mean Values of the Results of table I means mean value of the results of five iterations.

As evidenced by the results shown in table II, the MAE and RMSE values of the Resilient Backpropagation (RB) algorithm is the best among five neural network based algorithms experimented in the study with 80%, 0.05616 and 0.07046 as Accuracy, MAE and RMSE values respectively. The performance of Variable Learning Rate (VLR) and Variable Learning Rate training with momentum (VLRM) algorithms is not good as compared with Resilient Backpropagation algorithm. The performance of Batch Gradient Descent, Batch Gradient Descent with momentum algorithms in the study is not satisfactory with less than 50% Accuacy values in case of both algorithms.

TABLE I RESULTS OF DIFFERENT NEURAL NETWORK BASED ALGORITHMS FOR IDENTIFICATION OF REUSABLE MODULES

	Iteration	*Accuracy*	*MAE*	*RMSE*
BGD	1st	30	0.1560	0.1797
	2nd	20	0.2289	0.2661
	3rd	30	0.1743	0.1962
	4th	50	0.1670	0.2518
	5th	30	0.1379	0.1629
BGDWM	1st	40	0.1309	0.1679
	2nd	60	0.1299	0.2117
	3rd	40	0.1040	0.1292
	4th	50	0.1436	0.1914
	5th	30	0.1248	0.1398
VLR	1st	60	0.0841	0.1225
	2nd	40	0.1016	0.1353
	3rd	40	0.1336	0.1541
	4th	60	0.1033	0.1665
	5th	60	0.0627	0.0785
VLRM	1st	60	0.0929	0.1162
	2nd	70	0.0777	0.1056
	3rd	60	0.0680	0.0868
	4th	60	0.0758	0.1052
	5th	70	0.0747	0.1006
RB	1st	70	0.0579	0.0709
	2nd	90	0.0437	0.0610
	3rd	90	0.0550	0.0708
	4th	70	0.0737	0.0877
	5th	80	0.0505	0.0619

TABLE II MEAN VALUES OF THE RESULTS OF TABLE I

Algorithm	***Mean Accuracy***	***Mean MAE***	***Mean RMSE***
BGD	32	0.17282	0.21134
BGDWM	44	0.12664	0.168
VLR	52	0.09706	0.13138
VLRM	64	0.07782	0.10288
RB	80	0.05616	0.07046

IV. CONCLUSION

In this paper, different Neural Network based approaches are experimented to identify the reusability of function oriented software systems. We have used metric based approach for characterizing a software module. The metrics used are: McCabe's Cyclometric Complexity Measure for Complexity measurement, Regularity Metric, Halstead Software Science Indicator for Volume indication, Reuse Frequency metric and Coupling Metric. The neural networks experimented are: Batch Gradient Descent, Batch Gradient Descent with momentum, Variable Learning Rate, Variable Learning Rate training with momentum and Resilient Backpropagation. The Resilient Backpropagation (RB) algorithm is the best among five neural network based algorithms experimented in the study with 80%, 0.05616 and 0.07046 as Accuracy, MAE and RMSE values respectively. The performance of the Resilient Backpropagation (RB) algorithm is found to be consistent in all iterations that are recorded to calculate the mean result values. So, Resilient Backpropagation (RB) algorithm based approach can be used for the identification of the reusable component based on its structural properties as discussed in the paper.

The results obtained using proposed system is better than the results mentioned in literature .

The future work can be extended in following directions:

• This work can be extended to other programming languages.

• More algorithms can be evaluated and then we can find the best algorithm.

• Other dimensions of quality of software can be considered for mapping the relation of attributes.

REFERENCES

[1] Gill, Nasib S., "Importance of Software Component Characterization for Better Software Reusability", ACM SIGSOFT Software Engineering Notes, vol. 31 No. 1, Jan 2006, pp. 1-3.

[2] Gomes, P. and Bento, C., "A Case Similarity Metric For Software Reuse And Design", Artificial Intelligence for Engineering Design, Analysis and Manufacturing, vol. 15, issue 1, 2001, pp. 21-35.

[3] Isakowitz, T. and Kauffman, R.J., "Supporting Search For Reusable Software Objects", IEEE Trans. Software Eng., vol. 22, issue 6, Jun 1996, pp. 407-423.
[4] W. Lim, "Effects of Reuse on Quality, Productivity, and Economics," IEEE Software, vol. 11, no. 5, Oct. 1994, pp. 23-30.
[5] H. Mili, F. Mili and A. Mili, "Reusing Software: Issues And Research Directions," IEEE Transactions on Software Engineering, Volume 21, Issue 6, June 1995, pp. 528 - 562.
[6] G. Caldiera and V. R. Basili, "Identifying and Qualifying Reusable Software Components", IEEE Computer, February 1991, pp. 61-70.
[7] W. Humphrey, Managing the Software Process, SEI Series in Software Engineering, Addison-Wesley, 1989.
[8] L. Sommerville, Software Engineering, Addision-Wesley, 4th Edition, 1992.
[9] R. S. Pressman, Software Engineering: A Practitioner's Approach, McGraw-Hill Publications, 5th edition, 2005.
[10] G. Boetticher and D. Eichmann, "A Neural Network Paradigm for Characterising Reusable Software," Proceedings of the 1st Australian Conference on Software Metrics, 18-19 November 1993.
[11] Parvinder Singh Sandhu and Hardeep Singh, "Automatic Reusability Appraisal of Software Components using Neuro-Fuzzy Approach", International Journal Of Information Technology, vol. 3, no. 3, 2006, pp. 209-214..
[12] T. MaCabe, "A Software Complexity measure", IEEE Trans. Software Eng., vol. SE-2 (December 1976), pp. 308-320.
[13] G. Caldiera and V. R. Basili, Identifying and Qualifying Reusable Software Components, IEEE Computer, (1991), pp. 61-70.
[14] Herenji, H. R. and Khedkar, P (1992), "Learning and Tuning Fuzzy Logic Controllers through Reinforcements", IEEE Transactions on Neural Networks, vol. 3, 1992, pp. 724-740.
[15] Challagulla, V.U.B., Bastani, F.B., I-Ling Yen, Paul, (2005), "Empirical assessment of machine learning based software defect prediction techniques", 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems, WORDS 2005, 2-4 Feb 2005, pp. 263-270.