Framework for Evaluating Reusability of Procedure Oriented System using Metrics based Approach

Sonia Manhas
HOD, Information and Technology
SSCET, Badhani, Pathankot
Punjab

Rajeev Vashisht
Lecturer, Information and Technology
DAVIET, Jalandhar
Punjab

Reeta Bhardwaj
Lecturer, MCA Deptt
DAVIET, Jalandhar
Punjab

ABSTRACT
In this paper, we present the application of the neural network for the identification of Reusable Software modules in Procedure Oriented Software System. Metrics are used for the structural analysis of the different procedures. The proposed metrics for Procedure oriented paradigm are Cyclometric Complexity Using Mc Cabe's Measure, Halstead Software Science Indicator, Regularity Metric, Reuse frequency metric, Coupling Metric. The values of these Metrics will become the input dataset for the different neural network systems. Neural Network Based Approach is used to establish the relationship between different attributes of the reusability and serve as the automatic tool for the Evaluation of the reusability of the procedures by calculating the relationship based on its training. Different Eleven Training Algorithms of neural network are experimented and the results are recorded in terms of Accuracy, Mean Absolute Error (MAE) and Root Mean Square Error (RMSE). The results show that Conjugate Gradient with Powell/Beale Restarts (CGB) is best for the evaluation of reusable modules of procedure oriented software systems. Hence the proposed model can be used to improve the productivity and quality of software development.

Keywords
Software reusability, Neural Networks, MAE, RMSE, Accuracy, CGB.

1. INTRODUCTION
Software reuse is the important factor to enhance the improvement efforts of the productivity of the software because reuse can result in higher quality software at a lower cost and delivered within a shorter time [1]. Reused software is more accurate than new software because already it has been tried and tested in working systems. The initial use of the software reveals many design and implementation faults but when reused software are used then there are lesser number of faults. With the existence of the software there is less uncertainty in the cost of reusing which is an important factor for project management as it reduces the margin of error in project cost estimation. This is particularly true when relatively large software components such as sub-systems are reused. Reusing software can speed up system production because both development and validation time should be reduced. Thus the reuse of software in systems development is a strategy that increases productivity and quality.

Reuse is an act of synthesizing a solution to a problem based on predefined solutions to sub problems. The reuse activity is divided into six major steps performed at each phase in preparation for the next phase. These steps are:

- Developing a reuse plan or strategy after studying the problem and available solutions to the problem,
- Identifying a solution structure for the problem following the reuse plan or strategy,
- Reconfiguring the solution structure to improve the possibility of using predefined components available at the next phase,
- Acquiring, instantiating, and modifying predefined components,
- Integrating the components into the products for this phase, and
- Evaluating the products.

. 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 [2]. 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 [3]. 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 main purpose of procedure oriented software metrics is to predict the quality of the software modules. The various attributes that determine the quality of software modules are maintainability, understandability, readability, fault tolerance, reusability etc

Neural networks have seen an explosion of interest over the years, and are being successfully applied across a range of problem domains, in areas as diverse as finance, medicine, engineering, geology and physics. Indeed, anywhere that there are problems of prediction, classification or control, neural networks are being introduced. It can learn by example. In order to make a neural network useful, the user needs to gather representative data**,** and then invokes training algorithms to train the neural network.

Neural network learns about its environment through a set of input-output training samples and is an interactive process of adjustment applied to its synaptic weights and bias levels.

- The learning algorithm involves the following steps:
- The neural network receives the normalized inputs that are available in the input-output training data samples.
- The output of the artificial neural network is then computed.
- The output of the network is then compared with that given in the training data samples. The error in the output is computed by taking the difference of the desired output and computed output from the network.
- The synaptic weights and biases are then changed so as to decrease the error based on the error gradient with respect to the different synaptic weights.

The process is repeated until the desired error goal is achieved.

2. RELATED WORK

Normal One possible measure of a component's reusability comes from its success; how many other applications modules access this common code? Other measures come from static code metrics. There are basically two approaches to evaluate software reusability: qualitative and empirical. The qualitative methods require substantial manual effort, as these methods generally rely on a subjective value attached to how well the software adheres to some guidelines or principles. On the other hand, empirical methods depend on the objective data that can be collected automatically and cheaply with the help of some tool [4].

Selby [5] identified a number of characteristics of those components, from existing systems, that are being reused at NASA laboratory and reported that the developers were successful in achieving a 32 percent reusability index. Selby's recent experimental study has identified two categories of factors that characterize successful reuse-based software development of large-scale systems: module design factors and module implementation factors [6]. The module design factors that characterize module reuse without revision were: few calls to other system modules (i.e. low coupling), many calls to utility functions (i.e. high cohesion), few input-output parameters, few reads and writes, and many comments. The module implementation factors that characterize module reuse without revision were small in size (source lines) and with many assignment statements (i.e. low Cyclometric Complexity). The modules reused without revision had the fewest faults, fewest faults per source line, and lowest fault correction effort. The modules reused with major revision had the highest fault correction effort.

Reformat et al have used decision tree based approach to the problems of identification of good or bad software based on Java and C++ objects. In the study fifteen metrics have been used and 55 to 72% accuracy has been reported [7].

Prieto-Diaz and Freeman encouraged white-box reuse and identified five program attributes for evaluating reusability [8]. The attributes used are:

Program Size
Program Structure
Program Documentation
Programming Language
Reuse Experience

Chen and Lee developed about 130 reusable C++ components and used these components in a controlled experiment to relate the level of reuse in a program to software productivity and quality [9]. In contrast to Selby, who worked with professional programmers, Chen and Lee's experiment involved a team of 19 students, who had to design and implement small database system. The software metrics collected included the Halstead size, program volume, program level, estimated difficulty and effort. They found that lower the value of the software complexity metrics, the higher the programmer productivity [10].

Dunn and Knight also experimented and reported the usefulness of reusable code scavenging [11]. Chen, Nishimoto and Ramamoorty discussed the idea of subsystem extraction by using code information stored in a relational database [12]. They also described a tool called the C Information Abstraction System to support this process. Esteva and Reynolds [13] proposed the use of Inductive Learning techniques based on software metrics used to identify reusable modules. Their system was able to recognize reusable components.

Caldiera and Basili [14] proposed a tool called 'Care' that was used to identify reusable components according to a set of "reusability attributes" based on software metrics. The paper proposed four candidate measures of reusability based largely on McCabe and Halstead metrics. These attributes include measurement of utilization of the component in the problem domain, the cost of reuse and its quality. The 'Care' is expected to do the initial identification of the components having strong reusability characteristics; and then a domain expert will do a further examination of these components to determine their appropriateness to the domain, and package them to reuse.

Mayobre [15] described how these techniques can be extended and used to help in identifying data communication components of Hewlett-Packard.

Arnold [16] [17] mentioned a number of heuristics that can be used for locating reusable components in the Ada source code. The heuristics count the number of references to a particular procedure, identifying the loosely coupled modules and identifying modules that carry high cohesion.

The ESPRIR-2 project called REBOOT (Reuse Based on Object-Oriented Techniques) developed a taxonomy of reusability attributes. They listed Portability, Flexibility, Understandability and Confidence as four reusability factors. A list of criteria for each factor and metrics for each criteria, are also mentioned [18]. Although, some of the metrics depend on the subjective items such as checklists, an analyst can compute many of these metrics directly from the code. The analyst combines the individual metric values into an overall value of reusability.

The NATO Standard for the Software Reuse Procedures recommended tracking "Number of Inspections", "Number of Reuses", "Complexity" and "Number of Problem Reports" as indicators of software quality and reusability [19].

Hislop used function, form and similarity measures, to evaluate the software [20]. Torres and Samadzadeh [21] [22] conducted a study to determine the relation of information theory metrics and

reusability metrics. The study examined the effects of two information theory metrics, entropy loading and control structure entropy, on the software reusability and found that high entropy loading (Coupling) had a negative effect, while low control structure entropy (Complexity) had a positive effect on reuse.

3. METHODOLOGY OF WORK

Reusability evaluation System for Procedure oriented Software Components can be framed using following steps

3.1 Selection of Metric Suit for Procedure Oriented Paradigm

A framework of metrics is proposed for structural analysis of procedure or function-oriented software. The code of software is parsed to calculate the metric values. The following suits of metrics are able to explore different structural dimensions of procedure oriented components.

The proposed metrics for Function Oriented Paradigm are as follows:

- Cyclometric Complexity Using Mc Cabe's Measure [23][24]
- Halstead Software Science Indicator [23] [25]
- Regularity Metric [23][25]
- Reuse-Frequency Metric [23][25]
- Coupling Metric [23]:

Generate the values of these mentioned metrics which will become the input dataset of different neural network systems to evaluate the reusability.

3.2 Design & Evaluate Neural Network System

The following eleven Neural Network algorithms are experimented:

- Batch Gradient Descent
- Batch Gradient Descent with momentum
- Variable Learning Rate
- Variable Learning Rate training with momentum
- Resilient Backpropagation
- Scaled Conjugate Gradient
- Conjugate Gradient with Powell/Beale Restarts
- Fletcher-Powell Conjugate Gradient
- Polak-Ribiére Conjugate Gradient
- Levenberg-Marquardt
- BFGS Quasi-Newton

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

- Perform the training of the different neural networks with the training dataset.
- The trained Neural Network is evaluated against the testing data on the different comparison criteria as described in the next step.

3.3 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. The formula for calculating *MAE* is given in equation shown below:

$$MAE = \frac{|a_1 - c_1| + |a_2 - c_2| + \ldots + |a_n - c_n|}{n} \quad (1)$$

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. It is just the square root of the mean square error as shown in equation given below:

$$RMSE = \frac{|a_1 - c_1| + |a_2 - c_2| + \ldots + |a_n - c_n|}{n} \quad (2)$$

3.4 Conclusions Drawn

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

4. IMPLEMENTATION AND RESULTS

In this paper, the implementation of the algorithm is done in Matlab 7.0 environment and Neural Network toolbox for Matlab is used. The dataset of procedure oriented software is collected and Batch Gradient Descent, Batch Gradient Descent with momentum, Variable Learning Rate, Variable Learning Rate training with momentum , Resilient Backpropagation, , Scaled Conjugate Gradient, Conjugate Gradient with Powell/Beale Restarts, Fletcher-Powell Conjugate Gradient ,Polak-Ribiére

Conjugate Gradient ,Levenberg-Marquardt, BFGS Quasi-Newton based neural networks are experimented to obtain the results in terms of *Accuracy*, *MAE* and *RMSE* values. These neural networks are run with metric values as input and the tables I is showing the Results of different Neural Network Based algorithms for Identification of Reusable Modules in the function based software systems in terms of Accuracy, MAE, RMSE.

The result values of algorithms under study as shown in table-I depict that the *Accuracy*, *MAE* and *RMSE* values of the Conjugate Gradient with Powell/Beale Restarts(CGB) algorithm is the best among other neural network based algorithms experimented in the study with 90%, 0.0525 and 0.0556 as *Accuracy*, *MAE* and *RMSE* values respectively for procedure oriented software systems. The performance of Resilient Backpropagation(RB), Levenberg-Marquardt(LM), Fletcher-Powell Conjugate Gradient(CGF) ,Polak-Ribiére Conjugate Gradient(CGP) and BFGS Quasi-Newton(BFG) is not good as compared with Conjugate Gradient with Powell/Beale Restarts algorithm. The performance of Batch Gradient Descent, Batch Gradient Descent with momentum algorithms, Variable Learning Rate and Variable Learning Rate training with momentum in the study is not satisfactory with between 50%-65% Accuacy values in case of all these four algorithms.

Table I. Results of Various Neural Network Algorithms for Identification of Reusable Modules.

Algorithm	Accuracy	MAE	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	76	0.05616	0.07046
SCG	70	0.0681	0.0751
CGB	90	0.0525	0.0556
CGF	70	0.0586	.0772
CGP	80	0.0660	0.0839
LM	80	0.0558	0.0607
BFG	80	0.0589	0.0651

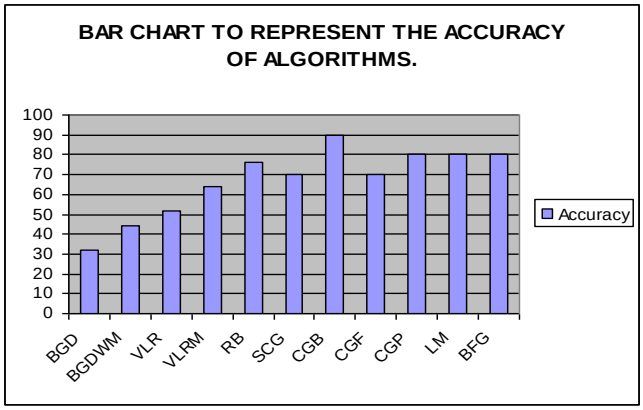

Fig. 1 Accuracy of Algorithms for Reusability Dataset

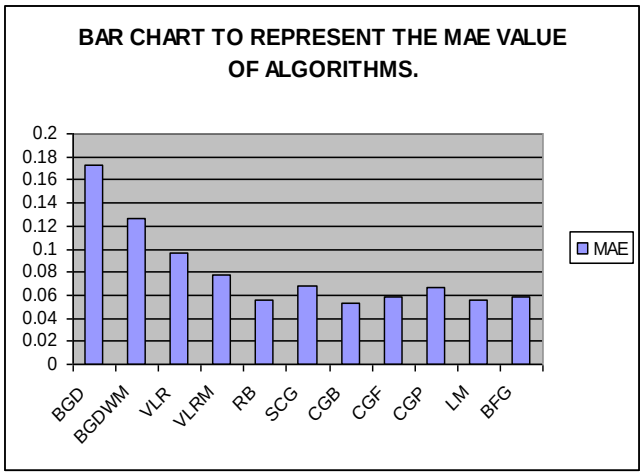

Fig. 2 MAE of Algorithms for Reusability Dataset

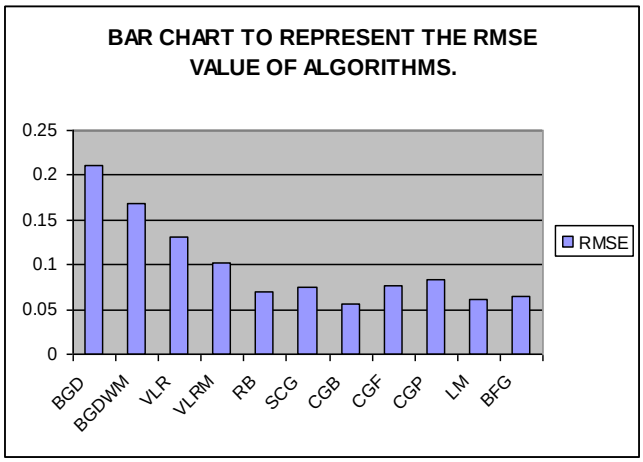

Fig. 3 RMSE Values of Algorithms for Reusability Dataset

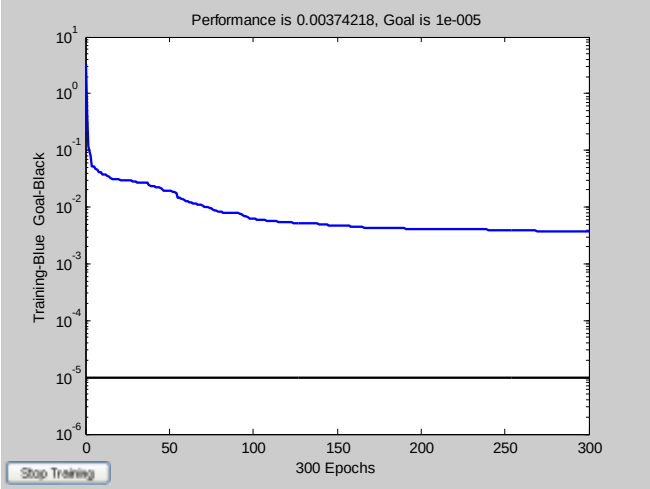

Fig.4 Training Performance of Conjugate Gradient with Powell/Beale Restarts (CGB) Algorithm for Reusability Evaluation

The graphical representation of the values of the performance results of the algorithms for the evaluation of the reusability value of procedure based software modules is shown in Fig. 1, Fig. 2 and Fig. 3 for the *Accuracy*, *MAE* and *RMSE* values respectively.

5. CONCLUSION

In this paper, eleven Neural Network based algorithms are experimented to develop the reusability evaluation system for procedure oriented software systems like c componenets. McCabe's Cyclometric Complexity Measure for Complexity measurement, Regularity Metric, Halstead Software Science Indicator for Volume indication, Reuse Frequency metric and Coupling Metric are used for structural analysis of a software module. Batch Gradient Descent, Batch Gradient Descent with momentum, Variable Learning Rate, Variable Learning Rate training with momentum, Resilient Backpropagation, Scaled Conjugate Gradient, Conjugate Gradient with Powell/Beale Restarts, Fletcher-Powell Conjugate Gradient ,Polak-Ribiére Conjugate Gradient ,Levenberg-Marquardt and BFGS Quasi-Newton algorithms are experimented. The algorithm Conjugate Gradient with Powell/Beale Restarts (CGB) is the best among eleven neural network based algorithms experimented in the study with 90%, 0.0525 and 0.0556 as *Accuracy*, *MAE* and *RMSE* values respectively. The performance of the Resilient Backpropagation (RB) algorithm is found to be best as compare to other algorithms that are recorded to calculate the mean result values. So, Resilient Backpropagation (RB) algorithm based approach can be used for the Modeling of the reusable component based on metrics discussed in this paper.

It can be further to other programming languages using other metrics and also more algorithms can be experimented to find the best algorithm.

6. REFERENCES

[1] I. Jacobson, M. Griss, and P. Johnsson, 1997 Software Reuse, Architecture, Process, and Organization for Business Success. Addison-Wesley.

[2] W. Lim, 1994 "Effects of Reuse on Quality, Productivity, and Economics," IEEE Software, vol. 11, no. 5, pp. 23-30.

[3] G. Caldiera and V. R. Basili, 1991 "Identifying and Qualifying Reusable Software Components", IEEE Computer, pp. 61-70.

[4] Poulin, J. S., 1997 Measuring Software Reuse–Principles, Practices and Economic Models, Addison-Wesley.

[5] Selby, R. W., 1988 Empirically Analyzing Software Reuse in a Production Environment in Software Reuse: Emerging Technology, W. Tracz, ed., IEEE Computer Society Press,

[6] Selby, Richard W., 2005 "Enabling Reuse-Based Software Development of Large-Scale Systems", *IEEE Trans. on Software Eng.*, vol. 31, no. 6, pp. 495-510.

[7] Reformat, M., Prdrycz, W. and Pizzi, N. J., 2003 "Software Quality Analysis with use of Computational Intelligence", *Journal of Information and Software Technology*, 45, pp. 405-417.

[8] Prieto-Diaz, Ruben Freeman, P., 1987 "Classifying Software for Reusability", *IEEE Software*, vol. 4, no. 1, pp. 6-16.

[9] Chen, Deng-Jyi and Lee, P.J., 1993 "On the Study of Program Reuse using Reusable C++ Components", *Journal of Software System*, vol. 20, no. 1, pp. 19-36.

[10] Poulin, J. S., 1997. Measuring Software Reuse–Principles, Practices and Economic Models, Addison-Wesley,

[11] Dunn, M. F. and J. C. Knight, 1993. "Software reuse in Industrial Setting: A Case Study", Proceeding 13th International Conference on Software Eng., Baltimore, MA.

[12] Chen, Y. F., Nishimoto, M. Y., and Ramamoorty, C. V., 1990 "The C Information Abstraction System", *IEEE Trans. on Software Eng.*, vol. 16, no. 3.

[13] Esteva, J. C. and Reynolds, R. G., 1991 "Identifying Reusable Components using Induction", *International Journal of Software Eng. and Knowledge Eng.*, vol. 1, no. 3, , pp. 271-292.

[14] Caldiera, G. and V. R. Basili, 1991 "Identifying and Qualifying Reusable Software Components", *IEEE Computer*, pp. 61-70.

[15] Mayobre, G., 1991 "Using Code Reusability Analysis to Identify Reusable Components from Software Related to an Application Domain", Proceedings 4th Workshop on Software Reuse, Reston. VA.

[16] Arnold, R.S., 1990 "Heuristics for Salvaging Reusable Parts From Ada Code", *SPC Technical Report*, ADA_REUSE_ HEURISTICS-90011-N.

[17] Arnold, R.S., 1990 "Salvaging Reusable Parts From Ada Code: A Progress Report", *SPC Technical Report*, SALVAGE_ADA_ PARTS_PR-90048-N.

[18] Karlsson, Even-Andre, Sindre, G. and Stalhane, T., 1992 "Techniques for Making More Reusable Components", *REBOOT Technical Report*, #41.

[19] Poulin, J. S.,, 1997 Measuring Software Reuse–Principles, Practices and Economic Models, Addison-Wesley.

[20] Hislop, G. W., 1993 "Using Existing Software in a Software Reuse Initiative", Proceedings 6th Annual Workshop on Software Reuse (WISR'93), Owego, New York.

[21] Poulin, J. S., 1997 Measuring Software Reuse–Principles, Practices and Economic Models, Addison-Wesley.

[22] Torres, W. R. and Samadzadeh, Mansur H., 1991 "Software Reuse and Information Theory based on Metrics", Proceedings Symposium on Applied Computing, Kansas City, MO, April 3-5, , pp. 437-446.

[23] Parvinder Singh Sandhu and Hardeep Singh, 2006,"Automatic Reusability Appraisal of Software Components using Neuro-Fuzzy Approach", International Journal Of Information Technology, vol. 3, no. 3, pp. 209-214.

[24] T. MaCabe, 1976 "A Software Complexity measure", IEEE Trans. Software Eng., vol. SE-2 pp. 308-320.

[25] G. Caldiera and V. R. Basili, (1991), Identifying and Qualifying Reusable Software Components, IEEE Computer, pp. 61-70.

[26] 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.

[27] 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.