International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:8, 2007

Automatic Reusability Appraisal of Software Components using Neuro-fuzzy Approach

Parvinder S. Sandhu and Hardeep Singh

***Abstract*—**Automatic reusability appraisal could be helpful in evaluating the quality of developed or developing reusable software components and in identification of reusable components from existing legacy systems; that can save cost of developing the software from scratch. But the issue of how to identify reusable components from existing systems has remained relatively unexplored. In this paper, we have mentioned two-tier approach by studying the structural attributes as well as usability or relevancy of the component to a particular domain. Latent semantic analysis is used for the feature vector representation of various software domains. It exploits the fact that FeatureVector codes can be seen as documents containing terms -the idenifiers present in the components- and so text modeling methods that capture co-occurrence information in low-dimensional spaces can be used. Further, we devised Neuro-Fuzzy hybrid Inference System, which takes structural metric values as input and calculates the reusability of the software component. Decision tree algorithm is used to decide initial set of fuzzy rules for the Neuro-fuzzy system. The results obtained are convincing enough to propose the system for economical identification and retrieval of reusable software components.

***Keywords*—** Clustering, ID3, LSA, Neuro-fuzzy System, SVD

I. INTRODUCTION

THE demand for new software applications is currently increasing at the exponential rate, as is the cost to develop them. The numbers of qualified and experienced professionals required for this extra work are not increasing commensurably [1]. Software professionals have recognized reuse as a powerful means of potentially overcoming the above said software crisis [2], [3] and it promises significant improvements in software productivity and quality [4], [5]. 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 exists 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]. 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.

Tracz observed that for programmers to reuse software they must first find it useful [7]. Experimental results confirm that prediction of reusability is possible but it involves more than the set of metrics that are being used [8]. According to [9], in some sense, researchers have fully explored most traditional methods of measuring reusability: complexity, module size, interface characteristics, etc., but the ability to reuse a software also depends on domain characteristics. It means we should concentrate on evaluating the software in terms of its relevancy to a particular domain.

The contribution of metrics to the overall objective of the software quality is understood and recognized [10]-[12]. But how these metrics collectively determine reusability of a software component is still at its naïve stage. A neural Network approach could serve as an economical, automatic tool to generate reusability ranking of software [13]. 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. Fuzzy systems, on the other hand, require a thorough understanding of the fuzzy variables and membership functions, of the input-output relationships, as well as the good judgment to select the fuzzy rules that contribute the most to the solution of the application. As for the Fuzzy inference system there is a need of membership rules for fuzzy categories. It is difficult to deduce these membership rules with a given set of complex data. Neural nets and fuzzy systems, although very different, have close relationship: they work with impression in a space that is not defined by crisp, deterministic boundaries [14]. Neural network can be used to define fuzzy rules for the fuzzy inference system. A neural network is good at discovering relationships and pattern in the data, so neural network can be used to preprocess data in the fuzzy system. Furthermore, neural network that can learn new relationships with new input data can be used to refine fuzzy rules to create fuzzy adaptive system. With the objective of taking advantage of the features of the both, we proposed Neuro-Fuzzy based approach to identify reusable components in existing systems.

The Manuscript was submitted for review on April 15, 2006.

Parvinder S. Sandhu is Assistant Professor with Computer Science & Engineering Department, Guru Nanak Dev Engineering College, Ludhiana(Punjab)-141006 INDIA(Phone: +91-98555-32004; Fax: +91161-2490339; Email: parvinder.sandhu@gmail.com, parvsandhu@yahoo.co.in)

Hardeep Singh is Profssor and Head with Computer Science & Engineering Department, Guru Nanak Dev University, Amritsar (Punjab) INDIA.

In the second section we have given details of the methodology followed. Third and fourth section is devoted to LSA and Neuro-fuzzy architecture. Fifth section discusses the implementation and results obtained. In Last section conclusion is made.

II. METHODOLOGIES FOLLOWED

A two-tier approach is proposed for evaluation of the domain-specific reusable code. In the first Tier, we tried to find the Domain-Relevancy of the Component and in the second tier Neuro-fuzzy system can be used to evaluate the software component's code reusability by analyzing structural properties of the component.

A. *Domain-Relevancy Appraisal*

We proposed an approach that allows to automatically cluster feature-vector (FV) codes, extracted from different domnains, into meaningful categories. It exploits the fact that FV codes can be seen as documents containing terms - the idenifiers present in the components- and so text modeling methods that capture co-occurrence information in low-dimensional spaces can be used. The FV code derived descriptions are computed by Latent Semantic Analysis (LSA) using Singular Value Decomposition (SVD) technique. The FV code representation of clusters is used to find the domain-relevancy (DR-value) of the software components using simialrity analysis.

1) Construction of Feature Vector (FV) of Domains

Following steps are proposed to find the FV of the different domains using training software components:

a) Extract identifiers

First, extract all identifiers from Training software belonging to different domains. Identifiers include function names, constant names and variable-names used in the software. From identifiers, exclude reserved because they have no relation with software features. We also cut out comments. The reason is that amount and quality of comments in each software system vary widely, and many software systems have copyright notice or license terms in comments.

b) Create identifier-by-software matrix

c) Remove useless identifiers and perform Normalization.

Before performing LSA, remove identifiers that appear in only one software system, or in more than half of software systems. Identifiers appearing in only one software system are not meaningful in LSA. And, identifiers appearing in more than half of software systems are probably a general term. After that Normalization of the matrix is performed.

d) Apply LSA for decomposition and Dimensionality Reduction

e) Perform Cluster Analysis

Cluster analysis is performed to form the FV of the cluster or domain of the software that can be used in the calculation of the domain relevancy of software. The clustering results and FV of each software Domain are saved in excel database sheets for future use.

2) Estimating Domain Relevancy value (DR-value)

Following steps can be taken to calculate DR-value of a potential reusable Component:

a) Extract the structural features

Features are extracted from the potential reusable component and FV is formed as discussed in section-1a and FV is mapped according to occurrence matrix's keyword list using "folding-in" process [15].

b) Perform Similarity Analysis

Similarity analysis between FV of the potential reusable component and the FV of different domains is performed and the similarity vector tells the relevancy level with existing domains. Here we are taking the assumption that the input software might belong to a number of domains with different extent.

B. Structural Analysis

Structural Analysis of the query component is performed using Neuro-fuzzy Inference system to evaluate the software component's code reusability using following steps:

1) Deciding the set of Metrics that are able to express the structural attributes helpful in evaluating reusability of software. We have used following set of metrics:

- Cyclometric Complexity Using Mc Cabe's Measure [16] [17]
- Regularity Metric[6]
- Halstead Software Science Indicator[18]
- Reuse Frequency Metric[6].
- Coupling Metric[12] [17]

let

a_i = number of functions called and Data Coupled with function "i"

b_i = number of functions called and Stamp Coupled with function "i"

c_i = number of functions called by function "i" and Control Coupled with function "i"

d_i = number of functions Common Coupled with function "i"

Then total lack of coupling measure m_c for function "i" can be calculated as shown in (1).

$$m_c = \frac{K}{w_1 a_i + w_2 b_i + w_3 c_i + w_4 d_i} \quad (1)$$

As lack of coupling(m_c) decreases, there is decrease in understandability and maintainability, so there should be some maximum value of the coupling associated with a software component, beyond which the component becomes non-reusable i.e. there should be minimum value for the lack of coupling measure(m_c).

2) Parse the software code to generate the Meta information related to the Software Metrics decided in previous step.

3) Calculated values of the metrics are forwarded as input to the trained Neuro-Fuzzy inference Engine.

4) Neuro-Fuzzy inference system, which is already trained, will get the metric values from the earlier stages and estimate the reusability value of the software components based on the structural attributes.

Considering the reusability value generated in last step along with the DR-value, the component can be extracted and put into the Reusable Software Reservoir for future reuse.

III. LATENT SEMANTIC ANALYSIS

Latent Semantic Analysis (LSA) can be applied to induce and represent aspects of the meaning of words [19], [20]. LSA is a variant of the vector space model that converts a representative sample of documents to a term-by-document matrix in which each cell indicates the frequency with which each term (rows) occurs in each document (columns). Thus a document becomes a column vector and can be compared with a user's query represented as a vector of the same dimension.

LSA extends the vector space model by modeling term-document relationships using a reduced approximation for the column and row space computed by the singular value decomposition of the term by document matrix. SVD is one technique being used in LSA.

1) Singular Value Decomposition (SVD)

SVD is a form of factor analysis, or more properly, the mathematical generalization of which factor analysis is a special case [19]. It constructs an *n* dimensional abstract semantic space in which each original term and each original (and any new) document are presented as vectors. In SVD a rectangular term-by-document matrix X is decomposed into the product of three other matrices W, S, and P^T as shown in (2).

$$X_K = W \ S \ P^T \tag{2}$$

Where W is a orthonormal matrix and its rows correspond to the rows of X, but it has *m* columns corresponding to new, specially derived variables such that there is no correlation between any two columns; i.e., each is linearly independent of the others. P is an orthonormal matrix and has columns corresponding to the original columns but *m* rows composed of derived singular vectors. The third matrix S is an *m* by *m* diagonal matrix with non-zero entries (called singular values) only along one central diagonal. A large singular value indicates a large effect of this dimension on the sum squared error of the approximation. The role of these singular values is to relate the scale of the factors in the other two matrices to each other such that when the three components are matrix multiplied, the original matrix is reconstructed.

Following the decomposition by SVD, the k most important dimensions (those with the highest singular values in S) are selected as shown in (3). All other factors are omitted, i.e., the other singular values in the diagonal matrix along with the corresponding singular vectors of the other two matrices are deleted. The reduced dimensionality solution then generates a vector of n real values to represent each document. The reduced matrix ideally represents the important and reliable patterns underlying the data in X. It corresponds to a least-squares best approximation to the original matrix X [20].

$$X_K = W_K S_K P_K^T \tag{3}$$

The X_k matrix should now contain the major associational structure in the matrix and has left out the noise. In this reduced model, the overall pattern of term usage determines how close the documents will be located, regardless of the precise words in the documents [21].

Document query similarity can be measured by calculating the cosine between the document vectors, x_k and a query vector, q_k as shown in (4)-(6).

$$S = \tilde{q}^T \tilde{A} \tag{4}$$

Where

$$\tilde{A} = S_K^{1-\alpha} P_K^T \tag{5}$$

And according to [13], the query vector is projected into the same k-dimensional space by (6).

$$\tilde{q} = q^T W_K S_K^{\alpha} \tag{6}$$

The performance of queries generally improves as k increases, but will decrease past a threshold. It is possible for an SVD based system to locate terms which do not even appear in a document. Documents which are located in a similar part of the concept space (i.e. which have a similar meaning) are retrieved, rather than only matching keywords. By using a concept space, Polysemy and Synonymy based problems can be solved.

2) Data Clustering

Nearest-neighbor-based, agglomerative, hierarchical, unsupervised conceptual clustering is proposed to create a hierarchy of clusters grouping of software of similar semantic structure. Clustering starts with a set of singleton clusters, each containing a single software $d_i \in D$, where i =1, ..., N, where D equals the entire set of documents and N equals the number of all software. The two most similar clusters over the entire set D are merged to form a new cluster that covers both. This process is repeated for each of the remaining N-1 software components. A complete linkage algorithm is applied to determine the overall similarity of document clusters based on the document similarity matrix. Merging of document clusters continues until a single, all-inclusive cluster remains. At termination, a uniform, binary hierarchy of document clusters is produced.

IV. NEURO-FUZZY SYSTEM'S ARCHITECTURE

The fuzzy logic approach is beneficial for measuring the reusability of a software component as the conventional model based approaches are difficult to be implemented. Unfortunately, with the increase in the complexity of the problem being modeled and unavailability of the precise relationship among various constituents for measuring the reusability, has led to rely on another approach which is mostly known as neuro-fuzzy or fuzzy-neuro approach. It has the benefits of both neural networks and fuzzy logic. The neuro-fuzzy hybrid system combines the advantages of fuzzy logic system, which deal with explicit knowledge that can be explained and understood, and neural networks, which deal with implicit knowledge, which can be acquired by learning. According to [22], A fuzzy system can be considered to be a

parameterized nonlinear map, called *f*, which can be expressed as (7).

$$f(x) = \frac{\sum_{l=1}^{m} y^{l} \left(\prod_{i=1}^{n} \mu_{A_i^l}(x_i) \right)}{\sum_{l=1}^{m} \left(\prod_{i=1}^{n} \mu_{A_i^l}(x_i) \right)} \quad (7)$$

where y^l is a place of output singleton, if Mamdani reasoning is applied or a constant, if Sugeno reasoning is applied. The membership function $\mu_{Ai}{}^{l}(x_i)$ corresponds to the input x=[x_1, x_2, x_3,… x_m] of the rule l . The "*and*" connective in the premise is carried out by a product and defuzzification by the center-of-gravity method. Consider a Sugeno type of fuzzy system having the rule base

Rule1: If x is A1 and y is B1, then f1= $p_1x + q_1y + 1$
Rule2: If x is A2 and y is B2, then f2= $p_2x + q_2y + r2$

Let the membership functions of fuzzy sets Ai, Bi, i=1,2, be , μ_{Ai}, μ_{Bi} .
-Evaluating the rule premises results in $w_i = \mu_{Ai}(x) * \mu_{Bi}(y)$ where i = 1,2 for the rule rules stated above.
-Evaluating the implication and the rule consequences gives (8).

$$f = \frac{w_1 f_1 + w_2 f_2}{w_1 + w_2} \quad (8)$$

Let

$$\overline{w_i} = \frac{w_i}{w_1 + w_2} \quad (9)$$

Then f can be written as (10).

$$f = \overline{w_1} f_1 + \overline{w_2} f_2 \quad (10)$$

In the neuro-fuzzy inference system using a given input/output data set, we have constructed a fuzzy inference system (FIS) whose membership function parameters are tuned (adjusted) using stochastic gradient descent rule with momentum for the parameters associated with the input membership functions. The initial rule-base for the Neuro-fuzzy system can be obtained using of the ID3 decision tree Generation algorithm. As a result, the training error decreases, at least locally, throughout the learning process.

V. EXPERIMENTATION AND RESULTS

We developed deployable Component Object Model (COM) based Component, which is Microsoft's binary standard for object interoperability. The developed component's objects can be accessible through Visual Basic, C++, or any other language that supports COM. We collected sample data from various Reusable Repositories of C components then we ran the program for the 21 components. The Clustering module of the program produces occurrence matrix and clustering results as shown in fig. 1 and fig. 2 respectively.

	alloc.c	at_wini_2.0.c	DELAS_image.c	deviceaccess.c	driver_minix3.0.c	exec.c
NR_HOLES	0.071428571	-5.71504E-18	1.34939E-17	8.6876E-18	-4.17944E-18	1.431E-17
NIL_HOLE	0.071428571	1.53006E-17	1.66604E-17	2.67739E-17	1.22727E-17	4.813E-17
hole	0.357142857	1.73164E-17	-2.56998E-17	9.17397E-17	-2.37765E-18	3.7561E-17
PROTOTYPE	0.142857143	0.126760563	4.77488E-17	0.029411765	0.037037037	0.125
alloc_mem	0.071428571	-8.81677E-18	-2.94294E-18	1.7834E-17	1.6794E-17	0.025
struct	0.571428571	-5.43218E-18	8.03542E-18	0.088235294	2.53168E-17	0.05
del_slot	0.285714286	1.46953E-17	7.56406E-18	6.53447E-17	-1.4125E-18	4.8596E-17
free_mem	0.142857143	-2.95275E-17	-1.07422E-17	3.36864E-17	8.66E-18	0.075
merge	0.214285714	1.03708E-17	5.4954E-18	5.15254E-17	1.73524E-19	3.3937E-17
max_hole	0.071428571	3.09452E-18	1.822E-18	1.62837E-17	-2.28473E-19	1.0766E-17
phys_clicks	0.071428571	3.61757E-18	1.85708E-18	1.63688E-17	-4.32588E-20	1.1875E-17
mem_init	0.071428571	3.47637E-18	1.60657E-18	1.63302E-17	-4.69707E-19	1.1609E-17
holes	0.071428571	3.76172E-18	1.6426E-18	1.67043E-17	-1.77014E-19	1.2412E-17
time	0.071428571	3.71181E-18	-1.80006E-18	1.71334E-17	4.43341E-18	1.3356E-17
REG_DATA	1.1017E-18	0.007042254	-1.65356E-18	-1.03079E-18	-2.73793E-18	-3.2676E-18
REG_PRECOMP	1.07692E-18	0.007042254	-1.6239E-18	-9.08415E-19	-3.06131E-18	-3.4462E-18

Fig. 1 Snapshot of Occurrence Matrix formed after the SVD decomposition

A	B	C	D	E	F
Cluster:1	Cluster:2	Cluster:3	Cluster:4	Cluster:5	Cluster:6
GEOMETRY_image.c	at_wini_2.0.c	ladder-game.c	reader_parsing.c	USB_DIAG.C	memalloc.c
SHOWTEST_image.c	deviceaccess_OS.c			xt_wini_driver.c	
DELAS_image.c	exec_OS.c			driver_minix3.0.c	
	gexec_cluster.c				
	gexec_lib_cluster.c				
	gexecd_cluster.c				
	GREP2MSG_utility.C				
	INT_IO.C				
	SCAN.C				
	TASM2MSG_utility.C				
	TCDISPLY.C				
	TCOMMAND_g.C				

Fig. 2 Clusters formed using Hierarchical Clustering

Dendrogram plots the hierarchical tree information as a graph is shown in fig. 3, where the numbers along the horizontal axis represent the indices of the objects or components in the original data set and the links between objects are represented as upside-down U-shaped lines. The height of the U indicates the distance between the objects. This height is known as the cophenetic distance between the two objects or components.

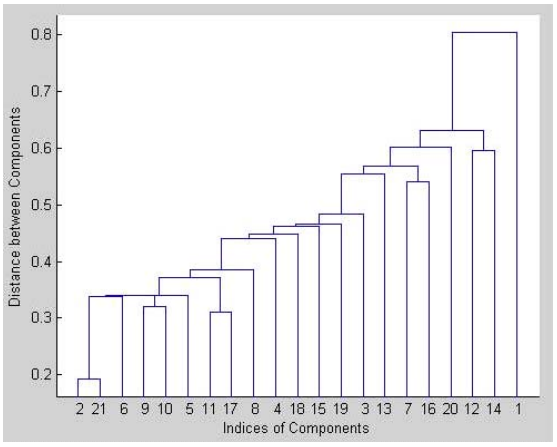

Fig. 3 Dendrogram showing the Distance between the Components

When the Domain-Relevancy module is run to determine DR-value of the input software component then the results of fig. 4 shows that the similarity level is highest with the 1st component already existing in the repository.

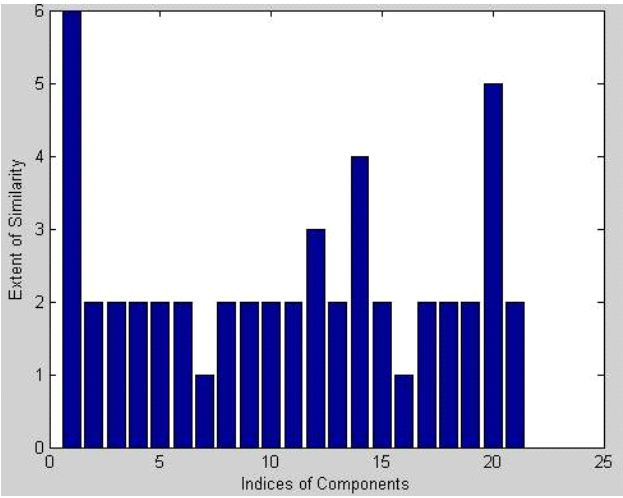

Fig. 4: Bar chart showing the extent of similarity of the input software with existing software

We tried to evaluate the system using Evaluation of precision and recall. Let *S* be a set of all software systems contained in a repository. Precision and recall are defined in (11)-(14).

$$\text{Precision} = \frac{\sum_{s \in S} precision_{soft}(s)}{|S|} \tag{11}$$

Where

$$precision_{soft}(s) = \frac{|C_{Actual}(s) \cap C_{Ideal}(s)|}{|C_{Actual}(s)|} \tag{12}$$

And

$$Recall = \frac{\sum_{s \in S} recall_{soft}(s)}{|S|} \tag{13}$$

Where

$$recall_{soft}(s) = \frac{|C_{Actual}(s) \cap C_{Ideal}(s)|}{|C_{Ideal}(s)|} \tag{14}$$

where $C_{actual}(s)$ is a set of clusters containing software "s", generated by our software and $C_{Ideal}(s)$ is a set of clusters containing input software "s", determined manually by the Domain Experts. Using Precision and Recall values we have calculated F-value as a measure of performance evaluation as shown in (15).

$$\text{F-Value} = \frac{2pr}{p+r} \tag{15}$$

Where *p* is the Precision and *r* is the Recall of the system. The best **F-Value for our system is 0.7**.

The Neuro-fuzzy approach based Inference Engine is implemented that takes values from the Metrics and calculates the Reusability Index of the Components after performing the domain-relevancy phase. A network-type structure similar to that of a neural network, which maps inputs through input membership functions and associated parameters, and then through output membership functions and associated parameters to outputs, can be used to interpret the input/output map is shown in the fig. 5.

The parameters associated with the membership functions will change through the learning process. The computation of these parameters (or their adjustment) is facilitated by a gradient vector, which provides a measure of how well the fuzzy inference system is modeling the input/output data for a given set of parameters. Once the gradient vector is obtained, any of several optimization routines could be applied in order to adjust the parameters so as to reduce some error measure. The Error Tolerance is used to create a training stopping criterion, which is related to the error size. The training will stop after the training data error remains within this tolerance. This is set to 0 as we don't know how training error is going to behave.

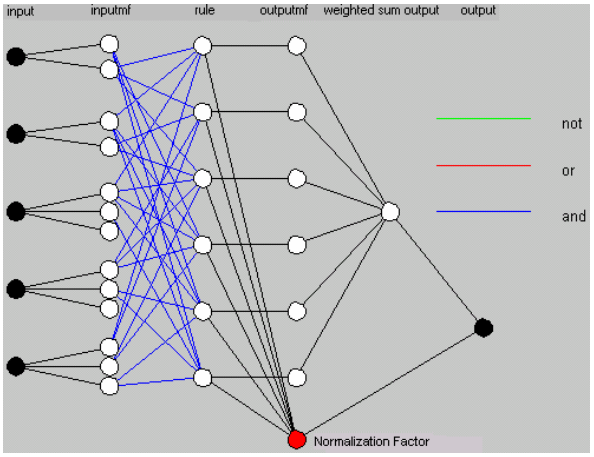

Fig. 5 Neural Network incorporating the fuzzy inference system

In the fuzzy Inference system Linguistic variables are then assigned to the input parameters based on their values. The assignment of the linguistic variables depends on the range of the input measurement.

Values to the linguistic variables of Complexity are assigned in terms of complexity of the software component. Cyclometric Complexity is assigned three linguistic variables "low", "medium" and "high" in the range of 0 to 10. The plot is also shown in fig. 6.

Values to the linguistic variables of Regularity are assigned in terms of level of regularity for the software component under consideration. Regularity is assigned two linguistic variables "yes" and "no" in the range of 0 to 1. The plot is also shown in fig. 6.

Values to the linguistic variables of Volume are assigned in terms of volume of the software module. Quality attribute Volume is assigned three linguistic variables "low", "medium" and "high" in the range of 0 to 10. The plot is also shown in fig. 6.

Values to the linguistic variables of Reuse-Frequency are assigned in terms of number of times the software module is reused. Reuse-Frequency is assigned two linguistic variables "Low" and "High" in the range of 0 to 10. The plot is also shown in fig. 6.

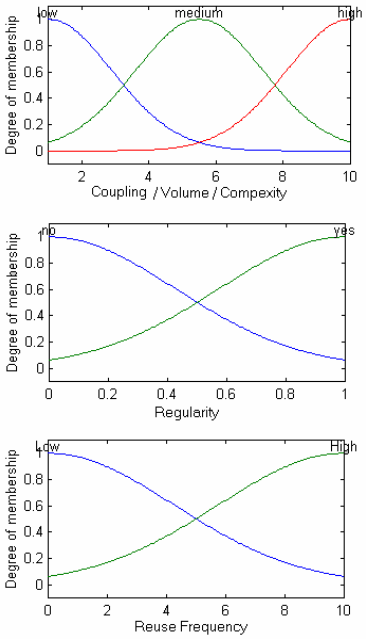

Fig. 6 Initial membership- functions for Fuzzy System

Values to the linguistic variables of coupling are assigned in terms of level of coupling of the software module with other modules or the level of dependency of the software module on other modules. Coupling is assigned three linguistic variables "low", "medium" and "high" in the range of 0 to 10. The plot is also shown in fig. 6.

Values to the linguistic variables of Reusability are assigned in terms of *"how reusable the software module is?"* As the output membership functions are only linear or constant for sugeno-type fuzzy inference. Reusability is assigned six linguistic variables PERFECT, HIGH, MEDIUM, LOW, VERY-LOW and NIL as constants in the range of 0-100.

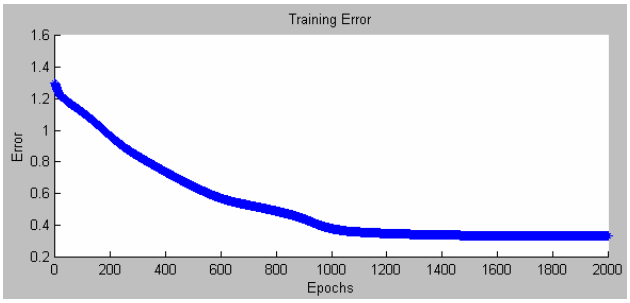

Fig. 7 Plot of Training error V/s Epochs

The training of the NEURO-FUZZY SYSTEM is performed using training data for 2000 iterations and the training error reduces after each iteration, as shown by the fig. 7 and stabilities at the **error value of 0.33591**, so at this point the network is said to be converged.

During the testing phase, when NEURO-FUZZY SYSTEM is tested against the testing data and the Average Testing error obtained equal to 0.41318. The plot between the actual testing output versus the expected testing output is shown in fig. 8.

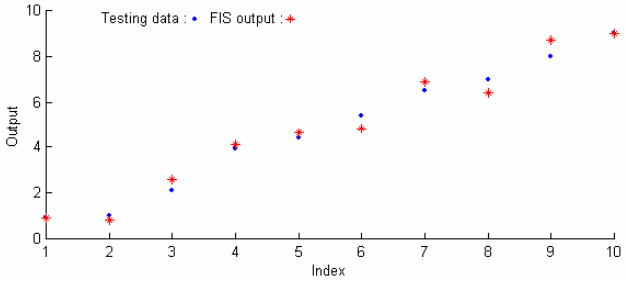

Fig. 8 Plot between the actual output and expected output for Testing data

VI. CONCLUSION

As the actual outputs produced by the Domain Relevancy module and neuro-fuzzy inference system are close to the expected output, so the above developed deployable COM based Component system, can be recommended for Automatic identification potential reusable components from the legacy systems and evaluating the quality of developed or developing reusable components for better productivity and quality.

ACKNOWLEDGMENT

The authors like to expree their gratitude towards Dr. S. B. Singh (Principal, G.N.D.E.C., Ludhiana) and Dr. H. K. Grewal, HOD (CSE & IT), G.N.D.E.C, Ludhiana for provision of laboratory facilities.

REFERENCES

[1] E. Smith, A. Al-Yasiri, and M. Merabti, "A Multi-Tiered Classification Scheme For Component Retrieval," *Proc. Euromicro Conference,* 1998, 24th Volume 2, 25-27 Aug. 1998, pp. 882 – 889.

[2] V.R. Basili, "Software Development: A Paradigm for the Future," Proc. COMPAC '89, Los Alamitos, Calif.: IEEE CS Press, 1989, pp. 471-485.

[3] B.W. Boehm and R. Ross, "Theory-W Software Project Management: Principles and Examples," *IEEE Trans. Software Eng.*, vol.15, no. 7, 1989, pp. 902.

[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. Tracz, "A Conceptual Model for Megaprogramming," SIGSOFT Software Engineering Notes, Vol. 16, No. 3, July 1991, pp. 36-45.

[8] Stephen R. Schach and X. Yang, "Metrics for targeting candidates for reuse: an experimental approach," ACM, SAC 1995, pp. 379-383.

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

[10] W. Humphrey, Managing the Software Process, SEI Series in Software Engineering, Addison-Wesley, 1989.

[11] L. Sommerville, Software Engineering, Addision-Wesley, 4th Edition, 1992.

[12] R. S. Pressman, Software Engineering: A Practitioner's Approach, McGraw-Hill Publications, 5th edition, 2005.

[13] 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.

[14] S. V. Kartalopoulos, Understanding Neural Networks and Fuzzy Logic-Basic Concepts and Applications, IEEE Press, 1996, pp. 153-160.
[15] T. Hofmann., "Probabilistic latent semantic indexing," In Proceedings of SIGIR'99, 1999.
[16] T. MaCabe, "A Software Complexity measure," IEEE Trans. Software Engineering, vol. SE-2, December 1976, pp. 308-320.
[17] Richard W. Selby, "Enabling Reuse-Based Software Development of Large-Scale Systems", IEEE IEEE Trans. Software Engineering, VOL. 31, NO. 6, June 2005, pp. 495-510.
[18] Maurice H. Halstead, Elements of Software Science, Elsevier North-Holland, New York, 1977.
[19] M. Berry, S.T. Dumais, and G.W. O'Brien, "Using Linear Algebra For Intelligent Information Retrieval," SIAM: Review, 37(4), 1995, pp. 573-595.
[20] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer and R. Harshman, "Indexing By Latent Semantic Analysis," Journal of the American Society For Information Science, 41, 1990, pp. 391-407.
[21] S. T. Dumais, "LSI meets TREC: A status report," Text Retrieval Conference, 1992, pp. 137-152.
[22] J-S. R. Jang and C.T. Sun, "Neuro-fuzzy Modeling and Control," Proceeding of the IEEE, March 1995.