Automatic Identification of Potential Reusable Mobile Components

Haeng-Kon Kim

Abstract—**The software can be developed from scratch or we can make use of already developed software components, which can enhance the productivity and quality. On Internet, a large collection of software code is being offered by open-access repositories but, how one can identify a relevant and good quality code with minimum effort? In this paper, the domain relevancy of software components appraised, by extracting the different aspects processed by those software components, with help of Probabilistic Latent Semantic Analysis. Further, structural attributes of software components are calculated using software metrics and quality of the software is inferred by Neuro-fuzzy Inference engine, taking these metric values as input. The neuro-fuzzy system is optimized by selecting initial rule-base through ID3 decision tree algorithm. The combination of DR-value with reusability value enables to identify a relevant or domain-specific and good quality code automatically. It has been found that Probabilistic Latent Semantic Analysis has provided better results than other retrieval techniques being followed. The reusability and domain-relevancy value determined is close to the manual analysis used to be performed by the programmers/repository managers. So, this kind of automation can improve the productivity and quality of software development.**

Index Terms—**Location Search, Reusable component, Domain specific, Component Based Development, Software Repository**

I. Introduction

In the present era of computerization there is utmost need of improving the productivity and quality of software as the demand of the software is increasing day by day. To achieve both the objectives it is always recommended to go for the software reuse that not only saves the time taken to develop the product from scratch but also delivers the almost error free code, as the code is already tested many times during its earlier reuse. But for an organization that has experience in developing software, but not yet used the software reuse concept, there exists the extra cost to develop the reusable components from scratch to build and strengthen their reusable software reservoir. 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 [1,2]. But the issue of how to identify reusable components from existing systems has remained relatively unexplored. Our approach to identification of reusable software is based on software models and metrics.

In literature, **Probabilistic Latent Semantic Analysis** technique is used for text-document similarity. In this paper, the domain relevancy of software components is determined, by extracting the different aspects processed by those software components, with help of Probabilistic Latent Semantic Analysis.

The contribution of metrics to the overall objective of the software quality is understood and recognized[3]. 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 [4,5]. When one designs with Neural Networks alone, the network is a black box that needs to be defined, this 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 [6]. 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[5], we make use of Neuro-Fuzzy based Inference System along with the domain relevancy appraisal system, to identify good quality and reusable components in existing object-oriented systems.

II. Related Works

A. Referential integrity

A system supports referential integrity if it guarantees that resources will continue to exist as long as there are outstanding references to the resources. The Web does not support this property and cannot do so since the system is unaware of the

Heang-Kon Kim is with the Department of Computer Engineering, Catholic University of Daegu Kyung San, Daegu, 712-702, Korea (corresponding author to provide phone: 053-850-2743; fax: 053-850-2740; e-mail: hangkon@ cu.ac.kr).

number of references that exist to a particular resource. It is impractical to maintain every resource that has ever been published on a particular server forever, this simply does not scale. Resources that are no longer of value, for whatever reason, become garbage and need to be collected. This may involve moving the resources to backing storage, or in some cases, deleting the resources entirely. Access pattern information, which is currently available through examination of server logs, is not a sufficient basis to decide whether an object is safe to garbage collect as important though rarely used references to a resource may exist. Safe garbage collection can only be performed if referencing information is available [7].

The consequences of deleting resources that are still referenced affects both the user and the information provider. Such broken links are the single most annoying problem faced by browsing users in the current Web. Broken links result in a tarnished reputation for the provider of the document containing the link, annoyance for the document user, and possible lost opportunity for the owner of the resource pointed to by the link.

B. Migration transparency

In addition to the problems associated with deleting Web resources, migrating resources (either intra- or interserver) also has the potential to break hypertext links. Using the URL naming scheme, when a resource moves its identity also changes. Therefore, hypertext links to the old name will now break, with the same consequences as stated previously. A partial solution to this problem is provided by the use of the HTTP redirect directive, which provides a forwarding pointer to the new location, allowing clients to rebind to the resource (automatically in the case of redirection-aware browsers). However, this is only a partial solution for the following reasons: firstly, documents containing references to the old location of the resource are not automatically updated, and so future requests will continue to access the old location first. Secondly, even if there were an automatic update mechanism, the lack of referential integrity means that the redirector can never be safely removed since it is impossible to determine whether all of the links have been updated. There is also the possibility that the URL may be reallocated following the migration[8]. The disadvantage of this approach is the performance penalty associated with name-server lookups, updates, and access bottlenecks. Furthermore, this scheme does not address the issue of referential integrity.

C. Resource and service management

As previously mentioned, most resource accesses through the Web are read-only operations. Updates to the resources by the information provider are performed using mechanisms orthogonal to the Web; that is, using the native commands and editing tools of the server machine. In effect Web resources reside in two distinct domains in parallel: the traditional structure of the file system and the complex interlinking Web of hypertext. Within these two environments, the interfaces to the

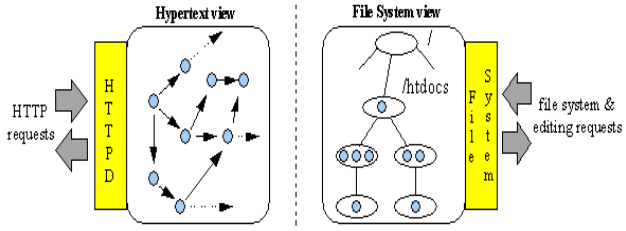

Fig.1. Disjoint interfaces to Web resources

resources as well as the relationships between the resources are fundamentally different, as illustrated in Figure 1.

Maintenance operations carried out by the information provider or site maintainer typically require manipulation of the resources within both domains. As described earlier in the section entitled "Migration Transparency," moving a resource within the file system has the side effect of changing its name in the Web domain. To reflect such changes, other Web resources must also be modified. Further, the internal state of the moved resource may also require modification, due to the use of relative naming [9]. At present such changes are performed manually and are prone to mistakes and inconsistencies. This can be seen as a result of having two parallel interfaces without any support for maintaining consistency.

D. Quality of service

The perceived quality of service (QoS) of the Web is influenced by many factors, including the broken link problems already mentioned. Even if a user holds a correct reference to an existing Web resource, it may still be unavailable due to a number of reasons, including unavailability of the machine serving the resource, and partitions in the network between the client and server. Partitions may either be real, caused by breaks in the physical network, or virtual, due to excessive network or server load making communications between the client and server impossible. Even if communication is possible, very poor response characteristics may effectively make the resource unusable. QoS will become more of an issue as the Web continues its transformation into a commercially oriented system

Technical solutions for improving QoS are fairly well understood, including caching for responsiveness, replication for availability, and migration for load balancing. Caching in the Web is reasonably common, both through the use of browser memory and disc caches, and also through the use of caching servers [10]. Effective caching is not a trivial task since there are many subtle problems that need to be addressed, including cache consistency, accounting, etc. Current caching servers use a heuristic approach for consistency management, where resources can only apply coarse-grained tuning based on expiry dates. URI working group is implementing a framework for resource replication, but appear to only be addressing the problem within the realm of read-only, static resources, where a read-from-any policy is appropriate. Replicating more complex read/write resources is much more difficult due to the problems associated with maintaining consistency between concurrent users and in the presence of failures.

E. Component repository

Component repository is a library system that supports to find, provide and manage components for building a business application. So it is a kind of tool that store, register and manage the all artifacts produced in component life cycle based on component architecture, and support a "Reuse with component" in CBD process through performing advanced retrieval and browsing of information. Most of all, component repository is a central mediator for component generation and utilization. So, analyzing and applying consistent meta data and user feedback information can establish CBD process including creation, verification, configuration management and circulation of component[11].

III. Neuro-fuzzy System's Architecture

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. 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.

A fuzzy system can be considered to be a parameterized nonlinear map, called f [5], which can be expressed as in equation (1).

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

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= $p_1x + q_1y + r1$

Rule2: If x is A2 and y is B2, then f2= $p_2x + q_2y + r2$

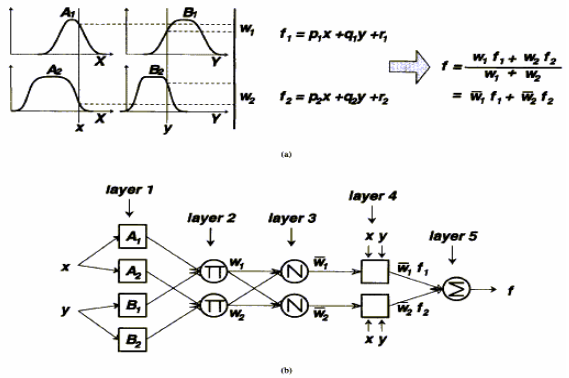

Fig.2. (a). A two-Input First-Order Sugeno Fuzzy Model with to rules (b). Equivalent Neuro-Fuzzy System

Let the membership functions of fuzzy sets Ai, Bi, i=1,2, be , μ_{Ai}, μ_{Bi} .

1. Evaluating the rule premises results in $w_i = \mu_{Ai}(x) * \mu_{Bi}(y)$ where i = 1,2 for the rule rules stated above.

2. Evaluating the implication and the rule consequences gives equations (1)-(4).

$$f = \frac{w_1 f_1 + w_2 f_2}{w_1 + w_2} \quad (2)$$

Let

$$\overline{w_i} = \frac{w_i}{w_1 + w_2} \quad (3)$$

then f can be written as

$$f = \overline{w_1} f_1 + \overline{w_2} f_2 \quad (4)$$

These all computations can be presented in a diagram form as shown in the figure 2(a) and 2(b).

A two-tier approach is proposed for evaluation of the domain-specific reusable code. In the first Tier, it is tried to find the Domain-Relevancy or usefulness 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

Following steps are proposed to find the DR-value of potential reusable components using training software components.

Training Phase

a)Extract identifiers from Training software belonging to different domains.

b) Create identifier-by-software matrix

c) Remove useless identifiers and perform Normalization to obtain n(d, w) matrix.

d) Initialize the P(w|z) and P(d |z) randomly with numbers between [0,1] and normalize them to sum to 1 along rows. P(z) is also initialize randomly.

e) Apply EM algorithm and iterate it until convergence or iterations are less than maximum number of iterations. The convergence means the maximization of log-likelihood function. The output of the Training phase is the probability of finding words in different latent classes, i.e. P(w|z) and probability of finding documents in different latent classes, i.e. P(d |z).

Estimating Domain-Relevancy value (DR-value)

a) Extract the features from q, the potential reusable components and FV is mapped according to occurrence matrix's keyword list.

b) Find different aspects' values in Query Software Components

After training, the estimated P(w|z) parameters are used to estimate P(q|z) for query software components, q , through a "folding-in" process. In the "folding-in" process, EM is used in a similar manner to the training process: the E-step is identical, the M-step keeps all the P(w|z) constant and only re-calculates P(q|z), which shows the level of different aspects in Query Software Components i.e. DR-value.

Cluster-analysis

Nearest-neighbor-based, agglomerative, hierarchical, unsupervised conceptual clustering can be used 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.

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:

Selection of Software Metrics

Following set of metrics are found to be true representative of different dimensions of structural attributes necessary for finding the quality of a reusable software component. Most of these metrics are also supported by recent findings of Selby in [8].

(1) Cyclometric Complexity Using Mc Cabe's Measure

According to Mc Cabe [12], the value of Cyclometric Complexity can be obtained using the following formula as shown in equation (5).

Cyclometric Complexity = Number of Predicate nodes + 1 (5)

Where predicate nodes are the nodes of the directed graph, made for the component, where the decisions are made i.e. predicate nodes should have more than one arrow coming out of it. If the complexity is low then reuse of component will not repay the cost. Otherwise high value of complexity indicates poor quality, high development cost, low readability, poor testability and prone to errors i.e. high rate of failure[3].

Hence the value of Cyclometric Complexity of a software component should be in between upper and lower bounds as an contribution towards reusability. If Cyclometric complexity is high with high regularity of implementation then there exists high functional usefulness.

Regularity

The notion behind Regularity is "How well we can predict length based on some regularity assumptions". As actual length (N) is sum of N1 and N2. The estimated length is shown in equation (6).

$$N' = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2 \quad (6)$$

The closeness of the estimate is a measure of the Regularity(r) of Component coding is shown in equation (7).

$$\text{Re}\,gularity = 1 - \frac{N - N'}{N} = \frac{N'}{N} \quad (7)$$

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.

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

C. Halstead Software Science Indicator

According to this metric volume of the source code of the software component is shown in equation (8).

$$\text{Volume} = (N_1 + N_2) \log_2 (\eta_1 + \eta_2) \quad (8)$$

Where n_1 = the number of distinct operators that appear in the program

n_2 = number of distinct operands that appear in the program

N_1 = The total number of operator occurrences

N_2 = The total number of operand occurrences

If the volume is high 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.

Reuse Frequency

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 are $X_1, X_2 \ldots X_N$ in the system, where $S_1, S_2 \ldots S_M$ are the standard environment components e.g. printf in C language.

$$\text{Re}\,use - Frequency = \frac{\eta(C)}{\frac{1}{M}\sum_{i=0}^{M}\eta(S_i)} \quad (9)$$

Equation (9) 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.

Coupling

Functions/methods that are loosely bound tend to be easier to remove and use in other contexts tan those that depend heavily on other functions or non-local data.

Different types of coupling effects reusability with different extent. Depending on the type of interface between two functions coupling can be classified in following categories:

- *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. In other words

functions share data items upon which control decisions are made.

- *Common Coupling*

In case of Common Coupling the two functions share global data items. Weight of coupling increases from "a" to "d" means Data coupling is lightest weight coupling, whereas Content Coupling is the heaviest one. 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 in equation (10).

$$m_c = \frac{K}{w_1 a_i + w_2 b_i + w_3 c_i + w_4 d_i} \qquad (10)$$

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).

IV. Implementation and Results

As a software implementation of the discussed concept, a deployable Component Object Model (COM) based Component, which is Microsoft's binary standard for object interoperability, is developed. The developed component's objects can be accessible through Visual Basic, C++, or any other language that supports COM. Sample data is collected from various Reusable Repositories of C components then we ran the program for the 62 components belonging to different domains and six unobserved latent variables are selected. The Training phase is run and P(w|z) is obtained as 718 * 6 dimensional matrix. Thereafter, 22 components are picked up as "q" and P(z|q) is calculated, as shown in figure 3. The figure shows different aspect or concept or unobserved latent variable levels in the software components and these values gives indication of the DR-values of the software component.

Bendrogram plots the hierarchical tree information as a graph is shown in figure 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.

I	J	K	L	M	N	O	P	Q	R	S
gexec_clusteringtem	GREP2MSG	pcmcia_diag.	SCAN.C	SHOWTEST	TCALC.C	TCDISPLY	TCOMMAND	TCPARSE	gphdraw.c	gtrdraw.c
0.702205431	0	0	0	0.681958668	0.846894873	0.876119	0.968871685	0.476258	0	0
0.611759233	0	0.986977139	0.940987	0	0.87641724	0.294303	0.777407198	0.049819	0	0
0	0	0.468742845	0	0	0	0	0.595821986	0.494903	0.889184	0.80676896
0.857134642	0.89141178	0	0.68931	0.945363127	0.896121269	0.948649	0.929675808	0.396216	0.9582892	0.800204121
0.775366933	0	0.557812518	0.508411	0	0.825821533	0.921569	0.931028876	0.691195	0.7514942	0.925141612
0	0.95553464	0	0	0.453646483	0	0	0.34147554	0.624737	0	0

Fig. 3. Snapshot of Calculated P(z |q) values

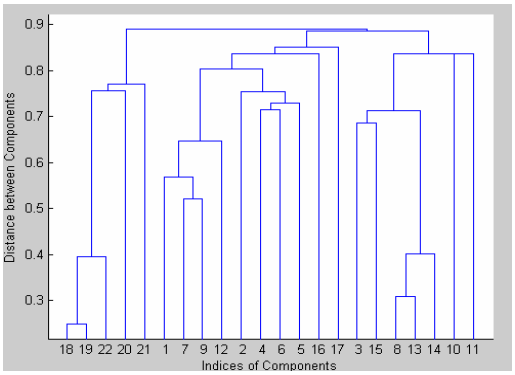

Fig.4. Bendrogram showing the Distance between the Components

Clustering analysis is performed on the P(z |q) and the software components are clustered in different clusters according to their latent variable values as shown in figure 4. Let's take the case of cluster:4, in which the components that are having concepts of "hardware device utility" and in cluster:6 most of the components contain strong "graphics concept".

There is a need to calculate value of the P(w|z), once with help of training data and later on, it can be used to calculate the DR-value of any number of potential reusable components.

It is 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 equations (11) to (15).

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

Where

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

And

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

Where

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

A	B	C	D	E	F
Cluster:1	Cluster:2	Cluster:3	Cluster:4	Cluster:5	Cluster:6
TCALC.C	pcmcia_diag.c	TCPARSER.C	alloc.c	USB_DIAG.C	gphdraw.c
TCDISPLY.C	gexec_clusteringtem	SCAN.C	at_wini_minixOS.c	DELAS_image.c	gtrdraw.c
TCOMMAND.C			deviceaccess_OS.c	GEOMETRY_ima	gxydraw.c
			driver_minixOS.c	SHOWTEST_ima	ncbi_memory_connector.c
			exec_OS.c		ncbi_socket_connector.c
			GREP2MSG_utility.C		

Fig.5. Result of the cluster analysis performed on P(z |q)

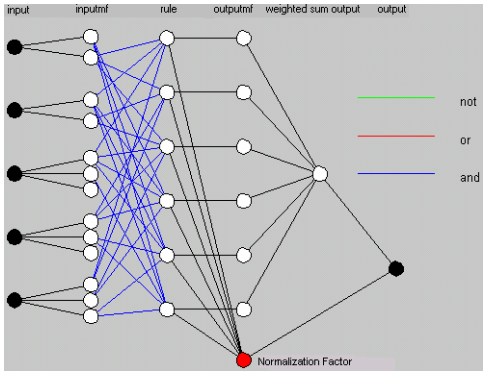

Fig.6. Neural Network incorporating the fuzzy inference system

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 i.e.

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

Where p is the Precision and r is the Recall of the system. The best F-Value for the system is 0.75.

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 figure 6. 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. 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.

During the testing phase, when NEURO-FUZZY SYSTEM is tested against the testing data and the Average Testing error obtained equal to 4.1318%. The plot for the tested data between the actual output versus the expected output is shown in figure 7.

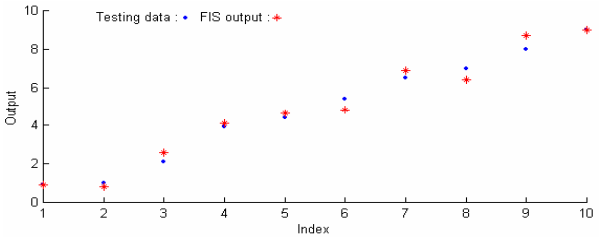

Fig.7. Plot between the actual output and expected output for Testing data

V. Conclusion and Future Works

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

Acknowledgment

" This work was supported by the Korea Science and Engineering Foundation (KOSEF) grant funded by the Korea government (MEST) (No. R01-2008-000-20607-0) "

References

[1] Dan Gildea and Thomas Hofmann, "Topic Based Language Models Using EM," Proceedings of 6th European Conference On Speech Communication and Technology (Eurospeech'99), 1999, pp. 2167-2170.

[2] G. Boetticher and D. Eichmann, "A Neural Network Paradigm for Characterizing Reusable Software," Proceedings of the 1st Australian Conference on Software Metrics, Nov. 1993, pp. 18-19.

[3] Guttorm Sindre, Reidar Conradi, Even-André Karlsson, "The REBOOT approach to software reuse," Journal of Systems and Software, vol. 30(3), Sep. 1995, pp. 201-212.

[4] G. Caldiera and V. R. Basili, "Identifying and Qualifying Reusable Software Components," IEEE Computer, February 1991.

[5] J-S. R. Jang and C.T. Sun, "Neuro-fuzzy Modeling and Control," Proceeding of the IEEE, Mar. 1995.

[6] M. H. Halstead, "Elements of Software Science," Elsevier North-Holland, 1977.

[7] Roger S. Pressman, "Software Engineering: A Practitioner's Approach," 5th ed. McGraw-Hill Publications, 2005.

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

[9] S. V. Kartalopoulos, "Understanding Neural Networks and Fuzzy Logic-Basic Concepts and Applications," IEEE Press, 1996, pp. 153-160.

[10] Thomas Hofmann, Jan Puzicha and M. Jorden, "Unsupervised Learning from Dyadic Data," Advances in Neural Information Processing systems, vol. 11, 1999.

[11] Thomas Hofmann, "Probabilistic Latent Semantic Indexing," Proceedings of the 22nd International Conference on Research and Development in Information Retrieval (SIGIR'99), Berkeley, 1999, pp. 35-44.

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

Dr. Haeng-Kon Kim is currently a professor in the Department of Computer Engineering and was a dean of Engineering College at Catholic University of Daegu in Korea. He received his M.S and Ph.D degree in Computer Engineering from Chung Ang University in 1987 and 1991, respectively. He has been a research staff in Bell Lab. in 1988 and NASA center 1978-1979 in U.S.A. He also has been reserched at Central Michigan University in in 2000-2002 and 2007-2008 in U.S.A. He is a member of IEEE, KISS and KIPS. Dr. Kim is the Editorial board of the international Journal of Computer and Information published quarterly by ACIS. His research interests are Component Based Development, Component Architecture & Frameworks for Mobile Applications and Components embedded systems .