A Neural Network based Method to Optimize the Software Component Searching Results in K-Model

Suresh Chand Gupta
PIET, Samalkha
Panipat, Haryana

Prof. Ashok Kumar
CSE Deptt., MMU
Mullana, Haryana

ABSTRACT
Here we propose a storage and retrieval approach of reusable software components based on UML diagram, metadata repository and neural network. If we search the repository on the basis of attributes of MDL file descriptions, the search result would be better and thus giving higher precision, as compared to keyword based search, then apply neural network to searching results of reusable software component for optimizing the searching results. The proposed approach is tested on various reusable software component datasets containing purely continuous or purely categorical or a mix of both types of attributes. Many features used in the analysis of reusable software component. In this paper reusable software component classified using feed forward back propagation Neural Network. One thousand sets of reusable software component obtained by software reusable techniques. The dataset consist of twenty eight features which represent the input layer to the FNN. The FNN will classify the reusable software component into type4, type3, type2 and type1 reusable software component. The sensitivity, specificity and accuracy were found to be equal 99.64%, 98.54% and 98.80% respectively. It can be concluded that FNN gives fast and accurate classification and it works as promising tool for optimizing the searching results of reusable software component. The overall accuracy of optimizing searching results of the proposed system is 96.50%. Thus, this approach is suitable for automated real time reusable software storing and searching.

General Terms
Software reuse, software component, Metadata, component retrieval, Component based engineering, Use Case Diagram, Class Diagram, MDL file.

Keywords
Metadata repository, UML Diagram, MDL File, Search Engine, K-model, ontology, neural network

1. INTRODUCTION
Software reuse [10, 24], the use of existing software artifacts or knowledge either partial, modified or complete, to create new software, is a key method for significantly improving software quality, reliability and productivity or in other words, it is the process of implementing or updating software systems using existing software assets. Software assets or components include anything that is produced from a software development effort. In this way a software component is developed only once, and can save out development effort multiple times. Many software developing organization uses CBSE as their software development standard because it reduces the development cost. To achieve this, CBSE is relying on reusability of software assets. So in directly it is clear that reusability is a key factor for reducing the development cost and an approach puts this idea central is called reuse-based software engineering [5] [14].

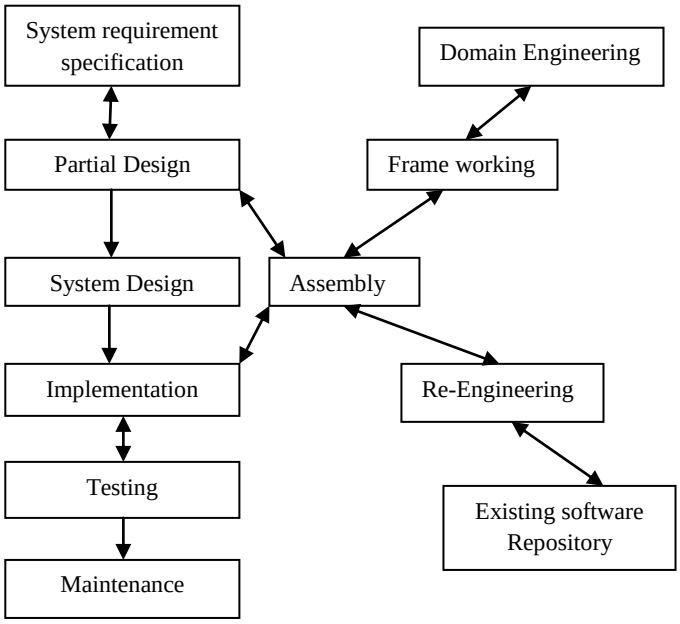
Software component reuse is an important concept to software development, as it reduces software development effort, time and cost and increase reliability, productivity, throughput and flexibility. Software component-Based Software Engineering proposes the reuse of software components, which can be retrieved and assembled into applications of specific domains [1, 2, 7]. In order to build these applications successfully, it is fundamental to choose appropriated software components from a collection of available software components. Thus, it is desirable to have a repository that supports the storage, query and retrieval of software components and makes reuse possible. Most existing software component repositories only retrieve a limited set of Software components and some do not satisfy user queries. Interrelated software components may exist and would be useful, but the user either does not know about them or is unable to retrieve them because the query is defined too narrowly [2] [4]. The schema of the repository itself often does not consider semantic relationships among software components and thus omits important component retrieval information. A technique to software component repositories is needed that provides the retrieval and recommendation of semantically interrelated software components. Reusing UML diagrams and source codes can help reduce development effort [7] [9].

A search engine can be developed that can help in reusing the existing UML components, software component design, source codes, test cases and maintenance case. While libraries of Unified Modeling Language (UML) diagrams and source codes do exist, one of the challenges that still remain is to locate suitable designs and source codes, and adapt them to meet the specific requirements of the software designer and developer. Traditional approaches to component retrieval are keyword-based; which resulted in the retrieval of many irrelevant components [7] [10]. A more promising approach is retrieval based on MDL format, where the contents of MDL file of the UML diagrams are matched to retrieve the software components. The UML models that are used for modeling are stored as MDL file format. These MDL file formats are semantically very information rich and contains lot of valuable information about the asset. The information can be structural as well as behavioral. The class diagram MDL file format contains valuable information about the structural description and contents of a class, i.e. class name, attributes, behavior, relationships, generalization etc. These attributes can be used for specification matching with the contents of the repository. The Use case diagram MDL file format contains valuable information about the requirements specification of

software. These include use cases and actors. If we search the repository on the basis of attributes of MDL file descriptions, the search result would be better and thus giving higher precision, as compared to keyword based search. Then apply neural network to improve the searching results [7] [19].

1.1 The K- Model

The K-model has been proposed as a alternative to address software reusability and reengineering practices during component-based software development. The creation of software is characterized by assembly and frame working. Although the main phases may overlap each other and iteration is allowed, the planned phases are: domain engineering, frame working, assembly, archiving, system analysis, design, implementation, testing, deployment maintenance and reengineering. The main characteristic of this software life cycle model is the emphasis on reusability during software creation and maintenance and the production of potentially reusable components that are meant to be useful in future software projects. Reusability implies the use of composition techniques during software development; this is achieved by initially selecting reusable components and assembling them, or by reengineering the existing software to a point where it is possible to pick out components. Frame working attempts to identify components and establish interrelationships within the application domain. Assembly focuses on selecting a collection of reusable frameworks or components from specific application domains. Reusability within this life cycle is efficient and more cost effective than within the traditional models because it integrates at its core the concern for reuse with assembling and the mechanisms to achieve it.



K-Model

The experience of using the K model has firstly shown that it is difficult to follow either a top-down or bottom-up approach so it is often necessary to switch over between them. It means that it is helpful to clarify high-level functionality for the software along with the identification of some low-level components and study their interrelationships. When developing large software, it is important to synthesize ideas from both top-down and bottom-up approaches. The idea of being able to classify parts of a software system as potentially reusable is a powerful concept. It is a time and cost saving formula and indeed more time can be spend on the specific aspects of the software than general aspects of the software that might be needed for specific application. The development of a component should therefore be with generality with reuse and reengineering in mind placing perhaps less emphasis on satisfying the specific needs of an application that is being developed. The specific parts of a design are those parts which turn a general set of components into a specific software system for a particular application. The K model supports "development with reuse and reengineering" through component assembly, and component archiving. Initially, the software engineer identifies potentially reusable components from existing reusable libraries. The components are then selected, adapted and reused through composition. For existing software the same technique can be used through using the concept of reengineering. At the end of software development, there may be many new reusable components that need to be verified, catalogued, classified and then stored into reusable libraries.

2. COMPONENT RETRIEVAL AND STORAGE

2.1 Software Component Retrieval using MDL File

The primary goal of this work is to identify a retrieval approach by using the MDL format. The UML models that are used for modeling are stored as MDL file format. These MDL file formats are semantically very information rich and contains lot of valuable information about the reusable software component. The information can be structural as well as behavioral. The class diagram MDL file format contains valuable information about the structural description and contents of a class, i.e. class name, attributes, behavior, relationships, generalization etc. The use case diagram MDL file format contains valuable information about the requirements specification of the software, i.e. it contains use cases, actors, and their relationships [7]. If we search the repository on the basis of attributes of MDL file descriptions, the search result would be better and thus giving higher precision, as compared to keyword based search. Moreover is we assign some numeric weights to different contents of a class, and arrange the search results in descending order, we would be able to find out the precision of the components in descending order. Then apply neural network algorithms to improve and optimize the reusable software component searching results. Hence the role of (re) user to find the best-fit component from the search result would be much easier. To successfully combine two paradigms software reuse and UML, for Component Retrieval, an automated tool must be designed, named as Component Retrieval Search Engine. The Purpose of Component Retrieval Search Engine is to retrieve best-fit or most reusable Component from the Repository as intended by the (re) user (developer). Moreover the search results to be displayed in descending order of percentage match with the input query. Hence the role of (re) user to find the best-fit component from the search results would be much easier. To model UML diagrams some software for modeling is required. In this work the software used is the Rational Rose, which is the product of IBM Corporation. In Rational Rose all the diagrams of the UML can be modeled and stored

in a single file of MDL file format. Hence various sample cases are taken and are modeled in Rational Rose. The repository should be indexed and classified according to the projects. Here our repository includes assets, namely source codes, designs and test cases. Designs are the diagrams, which are modeled in Rational Rose and stored in the Repository. Source codes consist of programs in any language. Designs and Source Codes about a project are stored in the Repository. The search engine should search upon the queries built to match the relevant components. The search input consists of MDL file. The output can be designs or source codes, as required by the user. The resultant should not only be most relevant matched items, but also some relevant matched items for browsing.

2.2 Software component storing and retrieval system

The function of a software component retrieval storing system is that construct the model of software component retrieval, in the model, functions, applied domains, work environments, working , static and dynamic behaviors of a software component can be accurately expressed, the software component can be store, searched and reused [4] [7]. A software component includes the entity, describing and metadata information in a software component repository. The three can be stored together or discretely. The discrete scheme is adopted so that reduce burthen, improve openness and is convenient for upgrade and maintenance, a component repository is divided into a describing repository and an entity repository. The software component retrieval system is based on meta-data, ontology faceted classification and adopts the model of three layers (view layer, application layer and data layer), the architecture is shown in Figure 1. The view layer is web form, the layer provides searching interfaces for software component users and library (repository) administration interfaces for administrators and knowledge experts. The application layer answer for describing component, classification, administration, feedback, authority and log, the layer realized by the view layer. There are four databases in data layer: a describing repository, component repository, a Meta data repository and ontology based component repository. The metadata repository stores information in special domains, provide accurate query terms, eliminate some phenomena such as same meanings with different names and same names with different meanings. According to describing facets, the describing repository can provide some information such as interfaces, functions, administrative levels, applied domains, developed languages, applied environments, editions and so on so that search software components[9]. The component repository store components and provide some services such as download and so on. More information on the domain semantics and also infer knowledge in order to recommend interrelated software components. The captured domain information should be related to the software components through an analysis of their purposes and functionalities. Thus, it is possible to relate software components to correspondent associations and entities in domain semantics. Therefore, the elements belonging to the meta-model permit retrieving and recommending components based on the analysis of semantic information. The software component retrieval is implemented based on the architecture of the software component retrieval system that is shown in Figure 1. A user input query terms with the interface of software component retrieval, these terms match terms in the metadata repository, and the fittest describing terms are chosen to feed back (if these terms cannot strictly match terms in the meta data repository, the thesauruses are chosen from the metadata repository by a heuristic algorithm [7] [10]), these terms are further filtered and refined by users so that accurate query describing terms is formed. An accurate requirement of users is reflected to a describing repository of software component based on faceted classification by a module of accurate query processing, appropriate software components will be searched by a fixed retrieval algorithm; users filter appropriate software components and download from the component repository of component. Then apply neural network algorithms to improve and optimize the reusable software component searching results. Hence the role of (re) user to find the best-fit component from the search result would be much easier. The whole searching process is shown in figure 1.

2.3 Searching Result Optimization using neural network

The objective of this study is to classifying reusable software component using feed forward back propagation neural network and Levenberg-Marquardt (LM) as the training algorithm. LM algorithm has been used in this study due to the reason that the training process converges quickly as the solution is approached. For this study, sigmoid, hyperbolic tangent functions are applied in the learning process. Feed forward back propagation neural network use to classify reusable software component according to reusable software component attribute and characteristic. FNN also classified reusable software component in type1, type2, and type3. Feed forward back propagation neural network is created by generalizing the gradient descent with momentum weight and bias learning rule to multiple layer networks and nonlinear differentiable transfer functions. Input vectors and the corresponding target vectors are used to train feed forward back propagation neural network. Neural network train until it can classify the defined pattern. The training algorithms use the gradient of the performance function to determine how to adjust the weights to minimize performance. The gradient is determined using a technique called back propagation, which involves performing computations backwards through the network. The back propagation computation is derived using the chain rule of calculus. The input vector is composed of 28 elements corresponding characteristic and attribute value of reusable software component. One hidden layers are determined empirically to be 16 and the output layer consists of 4 neurons. In addition, the transfer functions of hidden and output layers are tan-sigmoid and tan-sigmoid, respectively. For the training of neural network, the target is four element vectors.

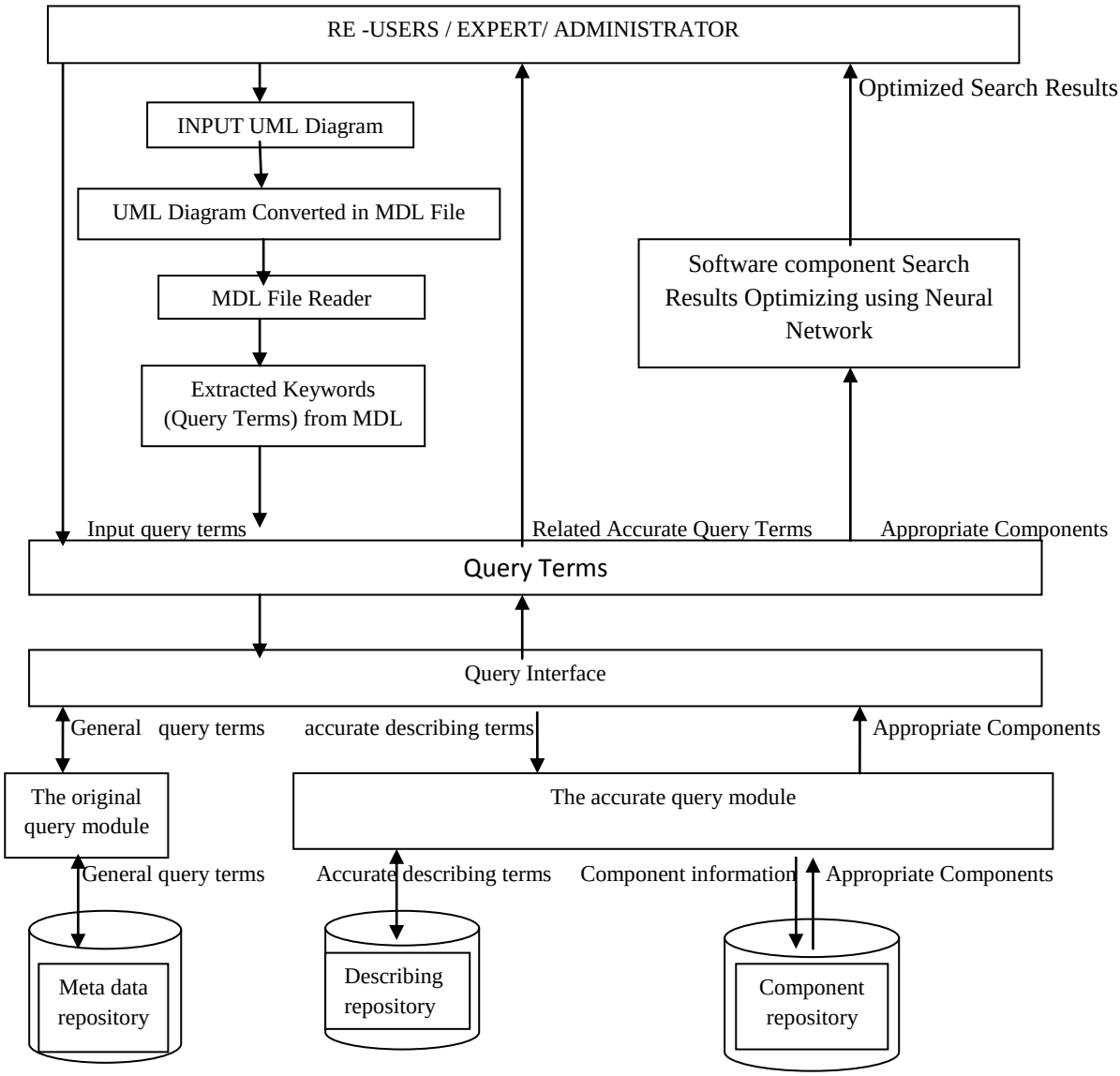


Fig 1: Proposed System for Reusable Software Components Storage and Retrieval

3. EXPERIMENT RESULTS

3.1 Neural Network Training and Testing Results

The proposed network was trained with feature vector data cases. When the training process is completed for the training data, the last weights of the network were saved to be ready for the testing procedure. The time needed to train the training datasets was approximately 28.60 minutes. The testing process is done for 1000 cases. These 1000 cases are fed to the proposed network and their output is recorded.

Performance plot: Performance plot show the training errors, validation errors, and test errors appears, as shown in the training process. Training errors, validation errors, and test errors appears, as shown in the following figure 2.

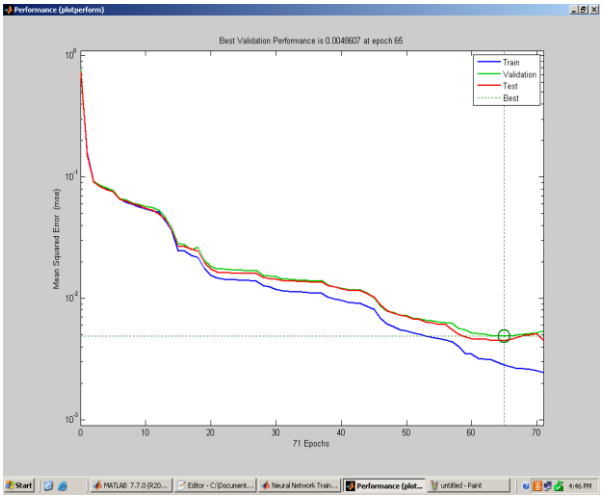


Figure 2: Performance plot

Receiver Operator Characteristic Measure (ROC) Plot: The colored lines in each axis represent the ROC curves. The *ROC curve* is a plot of the true positive rate (sensitivity) versus the false positive rate (1 -specificity) as the threshold is varied. A perfect test would show points in the upper-left corner, with 100% sensitivity and 100% specificity. For this problem, the network performs very well. The results show very good quality in the following figure 3.

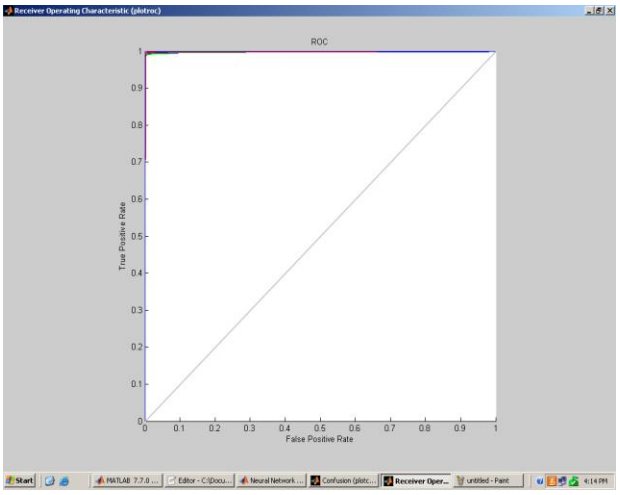


Figure 3: ROC Plot

Regression plots: This is used to validate the network performance. The following regression plots display the network outputs with respect to targets for training, validation, and test sets. For a perfect fit, the data should fall along a 45 degree line, where the network outputs are equal to the targets. For this problem the fit is reasonably good for all data sets, with R values in each case of 0.93 or above. The results show in the following figure 4.

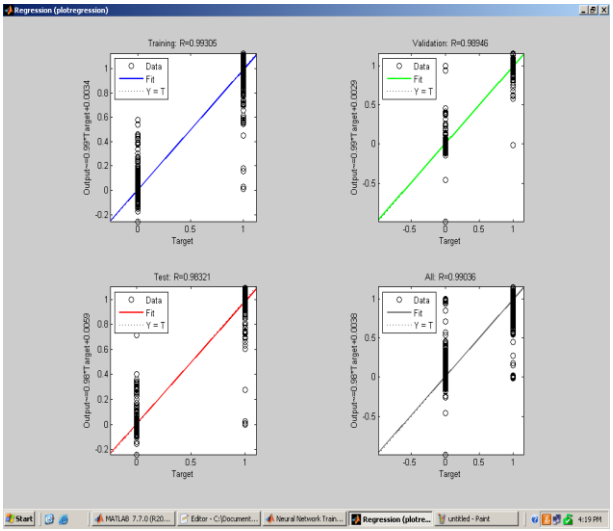


Figure 4: Regression Plots

Training State Plot: Training state plot show the deferent training state in training process and validation check graph. These plots also show the momentum and gradient graph and state in training process. The results show in the following figure 5.

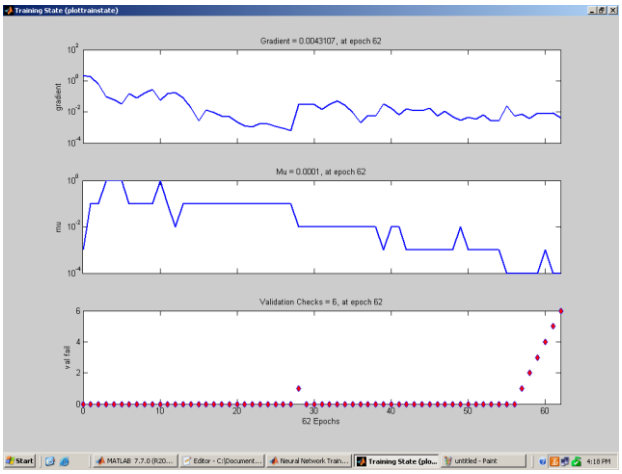


Figure 5: Training State Plot

Confusion Matrix: This figure shows the confusion matrices for training, testing, and validation, and the three kinds of data combined. The network outputs are very accurate, as you can see by the high numbers of correct responses in the green squares and the low numbers of incorrect responses in the red squares. The lower right blue squares illustrate the overall accuracies. The diagonal cells show the number of cases that were correctly classified, and the off-diagonal cells show the misclassified cases. The blue cell in the bottom right shows the total percent of correctly classified cases (in green) and the total percent of misclassified cases (in red). The results show very good recognition.

Figure 6: Confusion Matrix

Precision = Number of relevant components retrieved / Total number of components retrieved

Recall: Recall is defined as the number of relevant components retrieved divided by the total number of relevant components in the index.

Recall = Number of relevant component retrieved / Total number of relevant components in the index

Metadata, Component and UML diagram based Search results analysis:

Total components in the repository = 1000
Total number of components retrieved = 392
Total number of relevant components retrieved = 375
Total number of relevant components in the index =380
Precision =375 / 392 = 0.9566
Recall = 375/380 = 0.9868

- Attaining the precision of 0.9566 in Case 4 is considerably good which indicates that match is up to 96%.

Recall value of 0.9868 indicates that we would have been able to retrieve 99% relevant components, in Case 4

3.2 EXPERIMENT RESULT S ANALYSIS

The outputs of algorithms are depicted in the following figure 2, 3, 4, 5, 6, shows the reusable software component. Figure 6 show correctly classified reusable software component. The accuracy of classifier is defined as the ratio of the number of samples correctly classified to the total number of samples tested. The trained network has been tested in the retrieval mode, in which the testing vectors are not taking part in the training process. We have used the standard multilayered feed forward back propagation neural network trained using the gradient descent with momentum, resilient back propagation, and Levenberg-Marquardt algorithms. It produced 98.80% diagnosis accuracy respectively, where the 28 features of reusable software component are used as input of neural network. The overall accuracy of classification in the training, validation and testing mode are 99.64, 98.54 and 98.80%. The overall accuracy of classification show in the following figure 2, 3, 4, 5 and 6. Given these encouraging results, we are confident that an automatic reusable software component searching and storing system can be developed to assist the re user by providing second opinions and alerting them to component that require further attention.
Table 1, 2 and 3 show the results of proposed model used in the classification of reusable software component using neural network. The overall accuracy of classification in the testing mode is 98.80%.

Table1 .Performance results of reusable software component classification algorithm

Case study	Training accuracy %	Validation accuracy %	Testing accuracy %
Type 1 component	99.32	98.56	98.80
Type 2 component	99.20	98.58	98.82
Type 3 component	99.60	98.32	98.76
Type 4 component	99.20	98.50	98.82

Table 2 show total performance of the classification algorithm was evaluated by computing the percentages of Sensitivity (SE), Specificity (SP) and Accuracy (AC); the respective definitions are as follows:

$$SE=TP/ (TP+FN)*100 \quad (1)$$

$$SP=TN/ (TN+TP)*100 \quad (2)$$

$$AC= (TP+TN)/ (TN+TP+FN+FP)*100 \quad (3)$$

Sensitivity, specificity and accuracy of prediction have been calculated according to the above formals for all of the testing data (1000 reusable software component). Table 2 shows the resulted SE, SP and AC for testing data of the proposed networks.

Table 2.Performance results after tasting of the cancer tumor classification algorithm

No of component	Sensitivity	Specificity	Accuracy
1000	99.64%	98.54%	98.80%

The overall accuracy of classification in the training, validation and testing mode are 99.34%, 99.54% and 98.80%. The proposed reusable software component system gives fast and accurate reusable software component storing and searching.

Table 3: Experiment results

The Method of Retrieval	Components in Repository	Components Retrieved	Relevant Component Retrieved	Relevant Components in the Index	Precision	Recall
Traditional faceted retrieval	1000	380	280	350	73.68%	80%
MDL File based retrieval	1000	372	305	348	81.98%	87%
Metadata repository based retrieval	1000	375	325	356	86.66%	91.29%
Metadata repository and neural network based retrieval	1000	392	375	380	96%	98%

4 CONCLUSION AND FUTURE SCOPE

The suggested approach is tested on various reusable software component datasets from software component repository and results are encouraging. Components to be stored are developed such that these become more and more reusable. Making such a reuse library requires some different mechanism for storage and retrieval of software components. One such approach based on metadata, Component repository and neural network based searching was described in this paper with algorithm for building library and searching mechanisms. Combining software reuse with MDL file, metadata, and neural network is a new emerging trend in software development process. Combining these technologies helps the software development process by locating pre-existing components at the design time only, due to which the total effort of software development is decreased. It can be very difficult to decide reusable software component. FNN has been implemented for classification of reusable software component. The overall accuracy of classification in the training, validation and testing mode are 99.64, 98.54 and 98.80%. We are concluding that that the proposed system gives fast and accurate reusing software component. Given the encouraging test results, we are confident that an automatic software reusing system can be developed to assist the software developer and re user by providing second opinions and alerting them to component that require further attention.

5 REFERENCES

[1] F. Gibb, C. Mccartan and O. DonnellR, "The Integration of Information Retrieval Techniques within a Software Reuse Environment", Journal of Information Science, vol. 26, no. 4, pp.520- 539, 2000.

[2] W. Yuanfeng, Z. Yong and R.Hongmin, "Retrieving Components Based on Faceted Classification", Journal of Software, vol. 13, no. 8, pp.1546-1550, 2002.

[3] Lina and Z. Shijie, "Progress and prospects of expert system", Application Research of Computers, vol. 24, no. 12, pp.1-5, 2007.

[4] D. Hemer, "Specification-based retrieval strategies for component architectures", Proceedings of the 2005 Australian Software Engineering Conference (ASWEC'05), pp.233-242, 2005.

[5] R. Giliane, S. Luciana and H. Peter, "A Reference Model for Reusable Components Description", Proceedings of the 38th Annual Hawaii International Conference on Systems Sciences, Los Alamitos: IEEE Computer Society, pp.282-283, 2005.

[6] Li Ji-Dong, Xue-Jie Zhang and Yun-Shan Chen: "Applying Expert Experience to Interpretable Fuzzy Classification System using Genetic Algorithms," In Proc. 4th IEEE Int.Conf. on Fuzzy Syst & Knwldg Disc., vol. 02, pp. 129-133, Haikou, Hainan, China, Aug. 2007.

[7] Shekhar Singh,"An experiment in software component retrieval based on metadata and ontology repository", International Journal of Computer Applications (0975 – 8887), Volume 61– No.14, January 2013.

[8] Y. Wensheng, T. Pinghui and C. Xiuguo, "Problem Oriented Analysis and Decision Expert System with Large Capacity Knowledge-base", Proceedings of 2008 International Conference on Intelligent System and Knowledge Engineering, China, pp.32-37, 2008.

[9] Rajender Nath, Harish Kumar; Building Software Reuse Library; 3rd International Conference on Advanced Computing and Communication Technology- ICACCT-08; Asia Pacific Institute of Information Technology, Panipat , India; November 08-09, 2008, pp. 585-587.

[10] Rajesh K Bhatia, Mayank Dave, R.C Joshi, "A Hybrid Technique for Searching a Reusable Component from Software Libraries", DESIDOC Bulletin of Information Technology, Vol.27, No.5, September 2007, pp. 27-34.

[11] Rajesh K Bhatia, Mayank Dave, R.C Joshi, "Ant Colony Based Rule Generation for Reusable Software Component Retrieval", Proceedings of the 1st Conference on India Software Engineering Conference, pp 129-130, Feb 19-22, 2008, Hyderabad, India.

[12] Arun Sharma, Rajesh Kumar and P .S. Grover, "A Critical Survey of reusability aspects for component-based systems", Proceedings of World Academy of Science, Engineering & Technology, Vol. 21, Jan 2007.

[13] Clifton, C. and W. S. Li, "Classifying software components using design Characteristics", In proceedings of the 10th Knowledge-Based Software Engineering Conference, KBSE'95, IEEE Computer Society press, Los Alamitos, CA PP 139-146, 1995

[14] Daniel Lucredio, Antonio Francisico do Prado, Eduardo Santana de Almeida, "A Survey on Software Components Search and Retrieval", euromicro, pp.152-159, 30th EUROMICRO Conference (EUROMICRO'04), 2004

[15] Frakes,W.B and Pole,T, " An Empirical study of representation methods for reusable Software components", IEEE Trans. Soft Engg 20, 8,617-630, 1994

[16] Hafedh Mili, Fatma Mili and Ali Mili, "Reusing Software: Issues and research Directions," IEEE Transactions on Software Engineering, Vol. 21, No 6, 1995

[17] Henninger,S "An Evolutionary Approach to constructing effective software reuse Repositories", ACM Transactions on software engineering and methodology 6(2), 111-140, 1997

[18] Isakowitz,T and R,J Kauffman , "Supporting Search for Reusable Software Objects", IEEE Transactions on Software Engineering 22, 6, 407-423, 1996

[19] Jiang Guo, Lqui, "A Survey of Software Reuse Repostories", ecbs, p-92, 7th IEEE International Conference and Workshop on the Engineering of Computer Based Systems, 2000

[20] Jilani L L, R.Mili, M Frappier, J.Desharnais and A.Mili, "Retrieving Software Components that minimize adaptation effort", In Proceedings of the 12th IEEE International Automated Software Engineering Conference, ASE'97, IEEE Computer Society Press, Los Alamitos, CA pp 255-262, 1997a

[21] Jilani,L.L , R Mili and A Mili, " Approximate Retrieval: An Academic Exercise or a Practical Concern", In Proceedings of the 8th Annual workshop on software Reuse (WISR-8), 1997b

[22] Michail,A. & Notkin,D., "Assessing Software Libraries by Browsing similar classes, functions and relationships" , In Proceedings of 21st International Conference on Software Engineering (ICSE'99), ACM Press, Los Angeles, CA, pp. 463-472, 1999

[23] Mili R, Mili A and Mittermeir R.T, "Storing and Retrieving Software Components: A Refinement Based System", In Proceedings of 16th International Conference on Software Engineering, IEEE, pp.91-100, May 1994

[24] Mili and Edward Addy, Reuse Based Software Engineering (A Wiley-Interscience Publication, John Wiley and Sons, Inc.2002)

[25] Peter Eisinga and Jos Trienckens, Software Components for the Industry, From testing of applications to evaluation of components.

[26] Prieto-Diaz, "Implementing Faceted Classification for Software Reuse", Communication of the ACM 34, 5, 88-97, 1991

[27] Rajesh K Bhatia, Navneet Kaur, "Information Retrieval from a composite based Repository using Genetic Algorithms" 'IICAI 2005, page 667-675

[28] Rajesh K Bhatia, Mayank Dave, R.C Joshi, "Retrieval of most relevant reusable Component using genetic algorithms", Software Engineering Research and Practice 2006, 151-155

[29] Rajesh K Bhatia, Mayank Dave, R.C Joshi, "A Hybrid Technique for Searching a Reusable Component from Software Libraries", DESIDOC Bulletin of Information Technology, Vol.27, No.5, September 2007, pp. 27-34

[30] Rajesh K Bhatia, Mayank Dave, R.C Joshi, "Ant Colony Based Rule Generation for Reusable Software Component Retrieval", Proceedings of the 1st Conference on India Software Engineering Conference, pp 129-130, Feb 19-22, 2008, Hyderabad, India

[31] Rajiv D. Banker, Robert J Kauffman and Dani Zweig, "Repository Evaluation of Software reuse", IEEE Transactions on Software Engineering, Vol. 19, No 4, April 1993

[32] Rym Mili, Ali Mili and R.T.Mittermeir, "Storing and Retrieving Software Components: A Refinement Based System", IEEE Transactions on Software Engineering, Vol.23, No 7, July 1997