A Model Software Reuse Repository with an Intelligent Classification and Retrieval Technique

Article *in* Computer Science and Engineering · October 2012

DOI: 10.5923/j.computer.20110101.03

2 authors:

Priyanshul Niranjan
Motilal Nehru National Institute of Technology
16 PUBLICATIONS **52** CITATIONS

SEE PROFILE

Chakunta Venkata Guru Rao
SR Engineering College, Warangal, India
177 PUBLICATIONS **696** CITATIONS

SEE PROFILE

Computer Science and Engineering. 2011; 1(1): 15-21
DOI: 10. 5923/j.computer.20110101.03

A Model Software Reuse Repository with an Intelligent Classification and Retrieval Technique

P. Niranjan[1,*], C. V. Guru Rao[2]

[1]Computer Science and Engineering, Kakatiya Institute of Technology & Science, Warangal, 506015, India
[2]Computer Science and Engineering, S.R. Engineering College, Hasanparthi, Warangal, 506371, India

Abstract The essence of software reuse is the use of engineering knowledge or artefacts from existing software components to build a new system. Software reuse can significantly improve the quality of software products and reduces the overall development cost. Software reuse repository must be designed and developed in such a way that they can easily locate the components based on the requirements of the developers.
This work proposes a new methodology for efficient classification and retrieval of multimedia software components based on user requirements by using attribute classification scheme with genetic algorithm. In this intelligent classification we use Genetic algorithm that performs the classification of reusable software components in an intelligent manner and retrieves the components based on the requirements of the developers.

Keywords Software Reuse, Reuse Repository, Intelligent Classification, Genetic Algorithm

1. Introduction

Software reuse is the use of engineering knowledge or artefacts from existing software to build new systems[1]. The most common reuse product is the source code. Not only limiting to the source code reuse the other work products like the design, documentation, architecture, test data, tool and requirement specification can also be reused.

Software reuse is an important area of software engineering research that promises significant improvements in software productivity and quality[4]. Reuse has the potential to reduce cost, increase the quality of the products and shortens the time of software development. Reuse makes sense because the similarity found across software systems is significant. It is usually found that 60% to 70% of one development activity is common to the next activity. From this point of view software reuse can be promoted as a productivity and quality enhancement. As it is observed now a day's the cost pressure is increasing example in case of telecommunication and banking domains[3].

The biggest problem of software reusability in many organisations is the ability to locate and retrieve the existing software components. To overcome this problem, a necessary step is the ability to organize and catalogue collections of software components, to quickly search a collection to identify candidates for potential reuse, which can be used by developer to incorporate the components to build new efficient applications based on the requirements.

The best quality reuse repository tool is required to have a wide variety of high quality components, which are organized in an efficient manner using a classification technique and must be able to retrieve the best components that match user requirements. Effective software reuse requires that the users of the system have access to appropriate components. The user must access these components accurately and quickly and if necessary be able to modify them[2, 16].

This paper focus on the new methodology of intelligent classification and retrieval of software components from the reuse repository, this method implements a genetic algorithm for the effective classification of components in the repository and retrieves the best fit components from the repository based on the user requirements.

This paper is organized into the following sections. Section 2 is the literature survey which describes about the various existing classification techniques that are used to classify the components in the repository. Section 3 describes about the architecture of the proposed system in detail. Section 4 describes in detail about the intelligent classification and retrieval technique in two phases the component classification phase and retrieval phase. Section 5 explains about the genetic algorithm for identifying the classifiers. Section 6 deals with the experiments and the results. Section 7 explains about the graphs in details which deal with experimental results. Section 8 deals with conclusion and future work followed by references.

* Corresponding author:
npolala@yahoo.co.in (P. Niranjan)
Published online at http://journal.sapub.org/computer

2. Related Research

In the recent past research on software reuse has been focusing on several areas: examining programming language mechanisms to improve software reusability, developing software processes and management strategies that support the reuse of software, strategies for setting up libraries containing reusable code components and classification and retrieval techniques to help a software professional to select the component from the software library that is appropriate for his or her purposes.

2.1. Existing Software Component Classification and Retrieval Techniques

2.1.1. Free Text Classification

Free text retrieval performs searches using the text contained within documents. The retrieval system is typically based upon a keyword search[16]. All of the document indexes are searched to try to find an appropriate entry for the required keyword. The major drawback with this method is the ambiguous nature of the keywords used. Another disadvantage is that a search my result in many irrelevant components. A typical example of free text retrieval is the 'grep' utility used by the UNIX manual system. This type of classification generates large overheads in the time taken to index the material, and the time taken to make a query.

2.1.2. Enumerated Classification

Enumerated classification uses a set of mutually exclusive classes, which are all within a hierarchy of a single dimension[6]. A prime illustration of this is the Dewey Decimal system used to classify books in a library. Each subject area, e.g. Biology, Chemistry etc, has its own classifying code. As a sub code of this is a specialist subject area within the main subject. These codes can again be sub coded by author. This classification method has advantages and disadvantages pivoted around the concepts of a unique classification for each item. The classification scheme will allow a user to find more than one item that is classified within the same section / subsection assuming that if more than one exists. This type of classification schemes is one dimensional, and will not allow flexible classification of components into more than one place. As such, enumerated classification by itself does not provide a good classification scheme for reusable software components.

2.1.3. Attribute Value Classification

Attribute value Classification scheme uses a set of attributes to classify a Component[6]. For example, a book has many attributes such as the author, the publisher, and a unique ISBN number and classification code in the Dewey Decimal system. These are only example of the possible attributes. Depending upon who wants information about a book, the attributes could be concerned with the number of pages, the size of the paper used, the type of print face, the publishing date, etc.

2.1.4. Faceted Classification

Faceted classification schemes are attracting the most attention within the software reuse community. Like the attribute classification method, various facets classify components however there are usually a lot fewer facets than there are potential attributes. Ruben Prieto-Diaz[2, 8, 12 and 17] has proposed a faceted scheme that uses six facets. Each of the facets has to have values assigned at the time the component is classified.

3. Proposed System

Newly developed software components are classified and stored in reuse repository using the intelligent classification scheme. Existing components, which were classified using any one of the above four mentioned classification techniques, need to be reclassified according to the requirements. The architecture of the proposed system is shown below.

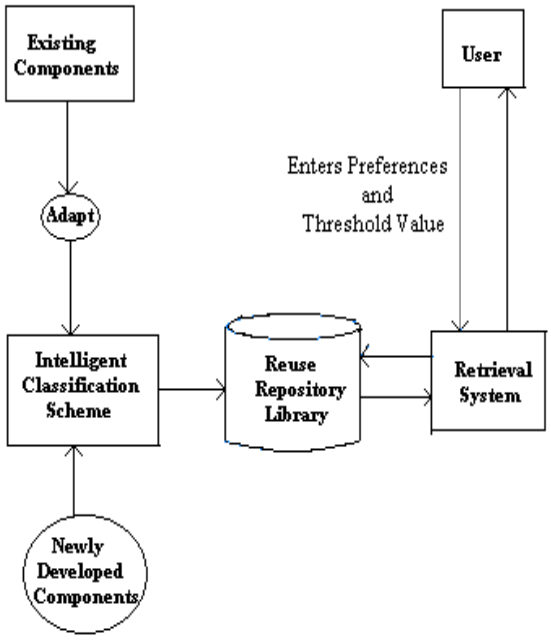

Figure 1. Proposed System Architecture.

The intelligent classification scheme describes each potential reusable component in the form of 9 characteristics. The classification scheme encodes the characteristics of the component into a series of bit strings (0's and 1's form) of 36 bits in length and these bit strings are processed by the genetic algorithm for the purpose of discovering the classifiers, where each classifier holds the homogeneous set of software components which are relevant in characteristics in its classifier set.

User will retrieve his desired component by specifying the required attributes of the components. The component retrieval system retrieves the desired components whose characteristics match the user specified attributes.

The following sections describes about the intelligent classification and retrieval techniques in detail.

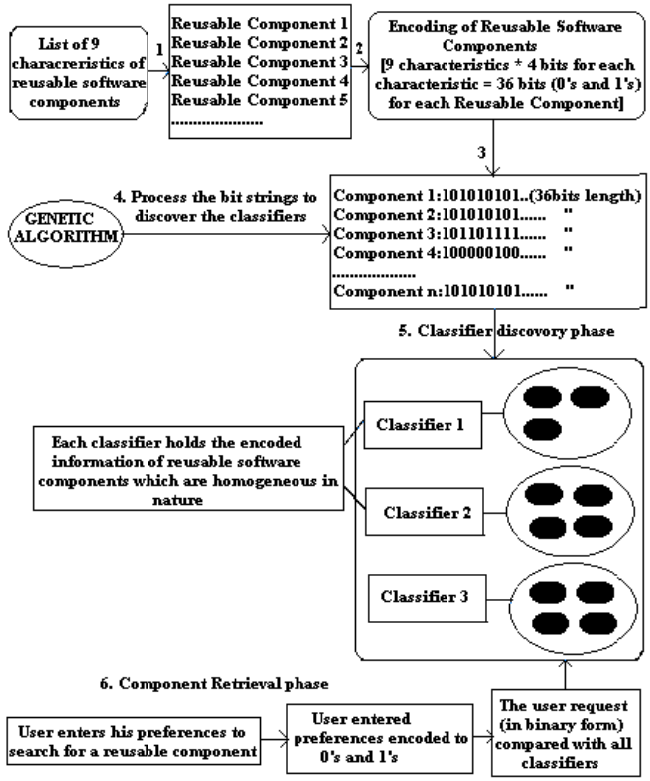

Figure 2. Detailed explanation of the intelligent classification and retrieval technique based reuse repository system.

4. Intelligent Classification and Retrieval Scheme

The Intelligent method of classification and retrieval of software components from the reuse repository is divided into two phases. The first phase is the component classification phase, which is sub divided into the encoding and classifier discovery phases. The second phase explains about the retrieval of components from the reuse repository.

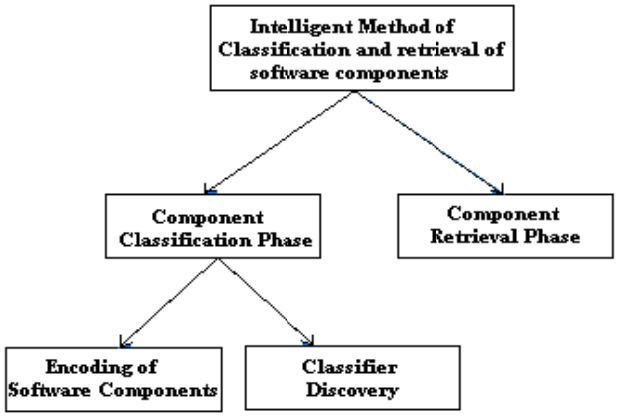

Figure 3. Phases of Intelligent Classification and Retrieval Technique.

4.1. Phase 1: Intelligent Component Classification Phase

In the component classification phase the components are classified using the 9 characteristics. The classification scheme uses a genetic algorithm which evolves the small number of classifiers by dividing the set of available components stored in the reuse repositories into certain subsets. This innovative way of classification of components using the genetic algorithm will result in the fast retrieval of the correct components according to the requirements that are specified by the user.

This phase is further divided into two sub phases, the encoding phase and the classifier discovery phase. In the encoding phase the components characteristics are encoded to binary form that is understood by the genetic algorithm. The genetic algorithm processes the bit strings and evolve the classifiers where each classifier holds the set of homogeneous software components.

4.1.1. Encoding of the Software Components

Each of the reusable software components are described with a set of 9 characteristics, which were identified by examining a component from both the functional and non-functional perspectives. The 9 characteristics include the following.

1. Component Name
2. Functionality
3. Domain
4. Operating System
5. Algorithm
6. Implementation Language
7. Developer
8. Time Complexity
9. Price

Each of the above mentioned characteristics of a software component are encoded to a binary string of 0's and 1's , where length of each characteristic of a component is four bits and therefore the total string length of the component is the aggregation of the number of bits needed to represent all possible values of each distinct characteristic. Therefore the total string length of each reusable software component is 36 bits, which includes all the 9 characteristics of a software component, represented in the binary form. This 36 bit string is used by the genetic algorithm to discover the component classifiers.

111011011100100000010011001010011010

Component Name:	**"Online Credit Card Transaction"**
Functionality:	**"Online Monetary Transaction"**
Domain:	**"Banking Domain"**
Operating System:	**Windows XP, Linux, Unix**
Algorithm:	**General**
Implementation Language:	**Java**
Developer:	**IBM**
Time Complexity:	**O(N)**
Price:	**$25-50**

Figure 4. Example of a Component encoding.

4.1.2. Classifier Discovery

The genetic algorithm attempts to discover several different classifiers, where each of the classifier, classifies a

number of software components into a homogeneous set in terms of characteristics. The classifier sets may have common elements as the classification process is based on component characteristics, with which it attempts to find large group of components with common values. There will be large no of components classified against a small number of classifiers.

Searching for a component will be performed by examining the user preferences against the classifiers rather than the actual components, something which will result in a fast searching process. The threshold parameter value specifies the similarity of a component with a classifier (that is the number of perfectly matched characteristics).

4.2. Phase 2: Component Retrieval

In the component retrieval phase, user will search for a specific component. First the user will enter the desired characteristics of a component which he wants to retrieve from the reuse repository, through an interface. Second the user will set the matching threshold value (obviously the lower the threshold value the more components will be returned and higher the threshold value, exactly the components that matches user entered characteristics will get returned).

The System will encode the user request as a bit string and will compare it against all classifiers that were discovered in the classifier discovery phase. The closest match will signify the "winning" classifier and the components that are classified under the winning classifier will get returned.

5. Genetic Algorithm for Identifying the Classifiers

A dedicated Genetic Algorithm[5] was developed to evolve candidate classifiers and select the optimal solution in terms of number of components in the corresponding classes, which works in discrete steps as follows:

1. Create a random population of 100 chromosomes (potential classifiers)
2. For every generation of genetic algorithm

2.1 Apply crossover operation to every pair of classifiers, where each pair is randomly selected according to the crossover probability

2.2 Apply mutation to a randomly selected classifier according to the mutation probability

3. Perform component classification for each of the 100 classifiers:

a) Compare each classifier's values of characteristics with those of each component. If component is close enough (determined by a threshold) to a classifier then assign the component to the class represented by this classifier.

b) Select the top 20 classifiers (chromosomes) in terms of the number of assigned components. Then find the average numbers of assigned components to 20 classifiers. This is the average fitness of current generation.

c) If the average fitness of the current generation is greater than that of the previous generation then create a new population by selecting chromosomes according to their fitness and repeat step 3. Otherwise do not create new population and repeat the step 2

The above algorithm is repeated until a termination condition is reached. In our case the algorithm terminates if no improvement in the average fitness of the population is observed for 100 generations. A very important parameter is the value of threshold, which determines whether a component belongs to a certain classifier. For example, a value of 40% means that at least 40% of the values of the classifier characteristics are identical to those of a component. This threshold value essentially determines the "success" level of a classifier to gather a rich number of components in his class.

6. Experimentation and Results

The first phase of the experiments was concerned with the classification of pool of components. In the second phase we investigate the retrieval of specific components.

6.1. Classification Phase

For the classification phase we created a randomly 1000 components, each comprising 36 bits. The results reported are averages over 100 runs. The classification of the components is based on the 9 characteristics as described in section 4. The threshold parameter is of paramount importance to our method, since it is a measure of similarity between the component characteristics and the classifier characteristics. We set the threshold value to assume the values of 30%, 40%, 50%, 60%, 70% and 80% for comparison purposes. The value of 30% produced classifiers, where each classified almost all of the available software components. This denotes that the classifiers derived cannot differentiate between the components. The threshold value of 80% did not produce good results either, because each classifier classified only between one and three components, which is also undesirable as it leaves many components unclassified.

The results for the threshold values of 40%, 50% and 60% are listed in Table 1. The "Average" column denote the average number of components classified by each classifier, while "Not Classified", denotes the number of unclassified components. The scores of 50% are quite successful, since there are no unclassified components and each classifier includes almost half of the components (47%). Thus, in the retrieval phase only half of the components need to be searched. Moving along the same line, the value of 60% is also satisfactory since each class contains a small number of components (58 on average), but there is a significant number of unclassified components. The threshold value of 40% did not perform at all as it classified almost all of the components are classified by its classifiers.

6.2. Retrieval Phase

In the Retrieval phase testing, we created a 10 random user requests searching for software components. Then the threshold value is set from 40% to 70% at increments of 10% as shown in Table2. We can observe that the 40% threshold returned a richer number of components, but not all of them were relevant to the user's requirements as expected. The 50% and 60% values retrieved less but more relevant components. The 70% threshold returned results for some queries only but it retrieved exact matching components for user's request.

Threshold Parameter Value: 40%		Threshold Parameter Value: 50%		Threshold Parameter Value: 60%	
Average (no. of Classified Components)	Average (no. of Unclassified Components)	Average (no. of Classified Components)	Average (no. of Unclassified Components)	Average (no. of Classified Components)	Average (no. of Unclassified Components)
920	0	450	0	50	270

Table 1. Experimental Results of Component Classification by 20 Classifiers

Threshold Parameter Value:40%	Threshold Parameter Value:50%	Threshold Parameter Value:60%	Threshold Parameter Value:70%
Average nuber of Components per Classifier			
27	20	12	2

Table 2. Retrieval Phase Results

7. Graphs

7.1. Graph for Comparing the Search Effectiveness of Various Classification Schemes

Search effectiveness refers to how well a given method supports in finding relevant components in the repository. It tells about the number of relevant items retrieved over the total number of retrieved items.

The graph for comparing the search effectiveness of various classification schemes is depicted below in figure 4. The horizontal axis on the graph represents the list of various existing classification schemes along with the intelligent classification scheme. The no of data items are represented along the vertical axis. The total data items retrieved are shown in white colour and the coloured area indicates the percentage of the relevant items among all the retrieved items.

Explanation:

Comparing with existing classification techniques, the integrated classification scheme performs well in retrieving the most relevant components according to the user requirements, but this scheme classifies components in the repository using only few attributes. Whereas, our proposed intelligent classification and retrieval system classifies the components in a broad manner on the basis of both functional and non-functional characteristics, which makes the proposed system more efficient in nature in retrieving most relevant components. The retrieved components working performance is highly commendable when integrated in the newly developing software systems, as the system supports the retrieval of most relevant items matching the user requirements.

The threshold value parameter which is important in the Intelligent classification and retrieval technique will effectively determine whether a component belongs to certain classifier or not. For example, a value of 40% threshold means that at least 40% of the values of the classifier characteristics are identical to those of a component. Thus, the selection of threshold value parameter will also plays a prominent role in retrieving most relevant components among the existing components.

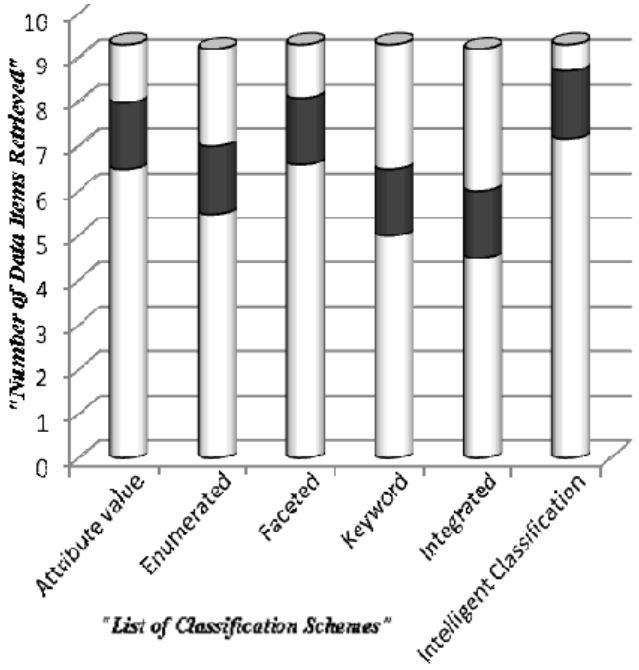

Figure 5. Finding Most Relevant Components.

The graph representing the finding of most relevant components is shown for the threshold values 40%, 50% and 60 % which marked the high performance results, in retrieving the most relevant components among all the components but they retrieved very less components but all of those matched with most of the user requirements.

7.2. Graph for Comparing the Search Time of Various Classification Schemes

Search time is the amount of time spent by the reuse repository system to locate the specific component in response to the user request. The graph for comparing search time of various classification schemes is shown in figure 5. The horizontal axis on the graph represents the various existing classification schemes along with the proposed scheme and the vertical axis represents the total search time to retrieve the components. The total data items retrieved are shown in white colour and the coloured area indicates the search time to retrieve those data items.

The intelligent classification and retrieval scheme uses a genetic algorithm, this genetic algorithm attempts to discover the several different classifiers, each of which classifies a set of homogeneous components in terms of charac-

teristics.

Searching for a component will be performed by examining the user preferences against the classifiers rather than the actual components this will result in a faster searching of components. The graph in figure 5 shows the search time of components in the proposed system. The graph showing the intelligent classification and retrieval scheme search time performs well for the threshold values of 50% and 60% with successful outcome.

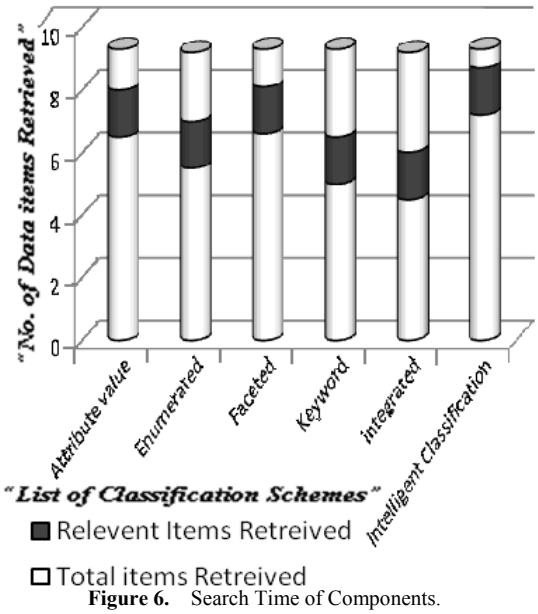

Figure 6. Search Time of Components.

8. Conclusion and Future Work

An effective software reuse repository software tool is designed and successfully implemented with the proposed intelligent classification and retrieval scheme. Our Classification is based on small set of classifiers which are evolved using the genetic algorithm. Each classifier evolved by the genetic algorithm attempts to classify the large number of software components according to the common characteristics. Retrieval of the relevant components is performed by comparing the user requirements with those of the classifiers. Thus, comparing a component's specification with only those of the classifiers instead of the entire set of available components in the repository will significantly save the search time of the components. A threshold is also used when evolving the classifiers, which determines the degree (percentage) of similarity with a classifier that is required to classify a component in a certain class. The threshold value has been found to have a profound influence in both the classifier's design phase (with the GA) and the retrieval phase.

Future work involved with this proposed intelligent scheme is the multimedia presentation of the components. Ranking of components that are returned by the system can also be included as an enhancement to future work.

REFERENCES

[1] William.B. Frakes and Kyo Kang, "Software Reuse Research Status and Future" IEEE transactions on Software Engineering, Vol. 31, No.7, July 2005.

[2] Ruben Prietzo-Diaz, "Implementing Faceted Classification for Software Reuse", Communication of the ACM, vol.34, no.5 May 1991.

[3] Title: "Reuse Repository" Ewa stemposz, Alina stasiecka, Kazimierz subieta Polish- Japanese Institute of Information Technology, Wersaw.

[4] William B. Frakes and Thomas. P.Pole, "An Empirical Study of Representation Methods for Reusable Software Components", IEEE Transactions on Software Engineering vol.20, no.8, Aug. 1994, pp.617-630.

[5] D. E. Goldberg, " Genetic Algorithms", Addison- Wesley, 1989.

[6] Jeffrey S. Poulin and Kathryn P.Yglesias "Experiences with a faceted Classification Scheme in a Large Reusable Software Library (RSL)", In The Seventh Annual International Computer Software and Applications Conference (COMPSAC'93), 1993,pp.90-99.

[7] Specification, Design and Implementation of a Reuse Repository, 31st annual international COMPSAC 2007, IEEE Transactions on Software Engineering, 2007.

[8] Ruben Prieto-Diaz, "Implementing Faceted Classification for Software Reuse" © 1990 IEEE, pp.300-304.

[9] P. Niranjan, C V Guru Rao"A Mock up tool for software component Reuse Repository" IJSE, vol 1 , no 2, April 2010.

[10] Chao- Tsun Chang, William c. chu, Chung-shyan Liu, Hongji Yang "A Formal Approach to Software Components Classification and Retrieval.

[11] Jain- Yun Nie, Francois Paradis, Jean Vaucher " Using Information Retrieval for Software".

[12] R. Prieto-Diaz and P.Freeman, "Classifying Software for Reuse", IEEE Software 1987, Vol.4, No.1, pp.6-16.

[13] Juan Llorens, Antonio Amescua, Manuel Velasco "Software Thesaurus: A Tool for Reusing Software objects", 1996 IEEE Transactions Proceedings of SAST.

[14] Aarthi Prasad, "AI- based Classification and Retrieval of Reusable Software Components" 1993 IEEE Transactions on Software Engineering.

[15] Achala Sharma, Daman Deep Kaur "Component Classification and Retrieval using Data Mining Techniques", Proceedings of National Conference on challenges & opportunities in Information Technology (COIT 2007).

[16] Gerald Jones and Ruben Prieto-Diaz, "Building and Managing Software Libraries", © 1998 IEEE, pp.228-236.

[17] Prieto-Diaz, Freeman, "Classifying Software for Reuse", IEEE Software, vol.4, mo.1, pp.6-16, 1997.

[18] Jung-eun cha, Young-jung yang, Mun-sub sung a,d Hang-gon kim, " Design and implementation of component repository for supporting the component based development process " IEEE 2001 paper.

[19] Mang Youuxin, Mong Xianghai, Yang Weimin "Component based Software Reuse Key Technology Research and Design", International Forum on Information Technology and Applications, 2009.

[20] B.Jalender, Dr.A.Govardhan, Dr.P.Preamchand "Breaking the Boundaries for Software Component Reuse Technology ",International Journal of Computer Applications, Vol 13, No.6, January 2011.

[21] Sarbijeet Singh, Sukhvinder Singh, Gurpreet Singh, "Reusability of the Software " International Journal of Computer Applications, Vol 7 , No.14, October 2010.

[22] Dr.C.V.Guru Rao, P.Niranjan "A Multilevel Representation of Repository for Software Reuse " , International Journal of Computer Science and Information Security, Vol 9 , No 9 , September 2011.