International Journal of Computer Science and Engineering Research and Development (IJCSERD), ISSN 2248-9363 (Print), ISSN 2248-9371 (Online), Volume 1, Number 2, May-October (2011) pp. 60- 69 © PRJ Publication http://www.prjpublication.com/IJCSERD.asp

IJCSERD

© PRJ PUBLICATION

AN EFFICIENT TOOL FOR REUSABLE SOFTWARE COMPONENT TAXONOMY

P.Shireesha
Assistant Professor,
KITS,Warangal
rishapakala@yahoo.co.in

Dr.S.S.V.N.Sharma
Professor,
Vagdevi College of Engineering, Warangal
ssvn.sarma@gmail.com

Dr.A.Govardhan
Principal,
JNTU, Kondagattu,
govardhan_cse@yahoo.com

ABSTRACT

Software Reuse acting as a prime role to develop applications in the organizations to improve productivity and quality. Promoting software reuse by developers largely depends upon the efficiency of the creator in describing them. The main difficulty in software reuse is the classification, storage, and retrieval of reusable components. Reuse demands that existing components must be readily incorporated into new products which enable efficient retrieval. The components must be suitably classified and stored in a repository to enable efficient retrieval. In this paper we developed an integrated classification scheme which uses a combination of existing classification schemes. A prototype of proposed system is developed by integrating two existing classification schemes.

Keywords: Software Reuse, Component Repository, Classification, Retrieval.

1. INTRODUCTION

Reuse is the use of previously acquired concepts or objects in a new situation, it involves encoding development information at different level of abstraction, storing this representation for future reference, matching of new and old situations, duplication of already developed objects and actions, and their adaptation to suit new requirements. Component is a well-defined unit of software that has a published interface and can be used in conjunction with other components to form larger units. Reuse deals with the ability to combine separate independent software components to form a larger unit of software. For example, in an auction application domain, one component captures the characteristics of a bid and its associated processes. Another component deals with transaction processing. These can be combined to form larger component that would be a reusable artifact. This kind of methodology greatly reduces the development time and cost. Thus it is a technique used to address the need for improvement of software development efficiency and quality as it involves the use of components from existing information systems to build new applications. However one problem is simply to

determine the proper scope for components. Few are of the opinion that all of the resources used on a project including human expertise should be reused. Others feel that reuse should focus on code, since this work is much more likely to have practical results. As such, any life-cycle product falls within the scope of reuse problem. Reuse then includes life-cycle objects as concept documents, estimates, requirements, designs, code, test plans, maintenance plans and user documentation. However out of this more emphasized area is software code.

The quest for software reusability paved the way for maintaining the reuse repositories comprising of reusable software artifacts extracted from already developed systems. The code snippets have to be classified, organized and stored in the library systems. These library systems are accessed by the users to retrieve the artifacts consistent with user requirements. Most of the software retrieval systems usually extract a set of reusable candidates ranked by similarity with user needs. However users don't want to invest a great deal of effort in selecting a component. So, in most cases, the list of retrieved candidates is not completely analyzed even is the highly ranked components are discarded. The user assumes that the best suited components are the ones on the top of the list of candidates. Thus to select a component from the list the user examines only the associated information for those first components. If none of them satisfies his requirements, may be he will try to refine or rewrite the original query or abandon the search. Therefore retrieval systems should exhibit more precision in their answers, by discarding some obviously unwanted components from the set of candidates and by retrieving only the ones that satisfy more precisely the requirements of the user. Software reuse is seen as the growing choice of the application programmers since past two decades. This is inference from the survey conducted to discover the needs and attitude of the developers towards reuse [1]. Most of them expect more from tools for automatic program generation. Research is ongoing to develop more user-friendly and effective reuse systems. A considerable number of tools and mechanisms for supporting reuse activities in software development have been proposed. They provide the assistance either to application developers for retrieving.

2. RELATED RESEARCH

Over approximately the past two decades, software reuse research has focused on several different area: examining programming language mechanisms to improve software reusability; developing software processes and management strategies that support the reuse of software; also, strategies for setting up libraries containing reusable code components, and classification and retrieval techniques to help a software engineer select the component from the software library that is appropriate for his or her purposes.

A classified collection is not useful if it does not provide the search-and-retrieval mechanism and use it.. A wide range of solutions to the software reuse classification and retrieval have been proposed and implemented. At different times, based on available software reuse systems and also based on researchers' criteria, software reuse classification and retrieval approaches have been classified differently, but with minor differences.

Ostertag et al. [18] classified the reported approaches by three types: 1) free-text keywords, 2) faceted index, and 3) semantic-net based. Free text based approaches basically use information retrieval and indexing technology to automatically extract

keywords from software documentation and index items with keywords. The free-text keyword approach is simple, and it an automatic process. But free-text keyword based approach is limited by lack of semantic information associated with keywords, thus it is not a precise approach. For faceted index approaches, experts extract keywords from program descriptions and documentation, and arrange the keywords by facets into a classification scheme, which is used as a standard descriptor for software components. To solve ambiguities, a thesaurus is derived for each facet to make sure the keyword matched can only be within the facet context. Faceted classification and retrieval has proven to be very effective in retrieving reuse component from repositories, but the approach is labor intensive. Semantic-net based approaches usually need a large knowledge-base, a natural language processor, and semantic retrieval algorithm to semantically classify and retrieve software reuse components. The semantic-net based approach is also labor intensive, and it is also rigid in narrow application domain.

Mili et al. [13] classifies search and retrieval approaches into four different types:

1) simple keyword and string match; 2) faceted classification and retrieval; 3) signature matching; and 4) behavior matching. The last two approaches i.e. signature matching and behavior matching are cumbersome and inefficient [4]. The classifications here for the first two approaches i.e. simple keyword and string match and faceted classification and retrieval are same as other researchers' classification of the two approaches.

Mili et al. [13] designed a software library in which software components are described in a formal specification: a specification is represented by a pair(S, R), where S is a set of specification, and R is a relation on S. The approach is classified as a keyword-based retrieval system, while matching recall is enhanced with sufficient precision: a match is considered as along as a specification key can refine a search argument. Besides there are two retrieval operations: exact retrieval and approximate retrieval. If there is no exact retrieval, approximate retrieval can give programs that need minimal modification to satisfy the specification.

The faceted classification scheme for software reuse proposed by Prieto-Diaz and Freeman [4] relies on facets which are extracted by experts to describe features about components. Features serve as component descriptors, such as the component functionality, how to run the component, and implementation details. To determine similarity between query and software components, a weighted conceptual graph is used to measure closeness by the conceptual distance among terms in a facet.

Vitharana et al. [10] proposed a scheme to classify and describe business components within a knowledge based repository for storage and retrieval. Two important steps in their proposed scheme are: 1) classification and coding for business components, 2) knowledge based repository for storage and retrieval. The classification groups similar parts, and symbols are coded. They borrowed the idea from facet-based scheme to describe features of reusable software artifacts, whereas their classification and coding scheme considers higher level business oriented features. In their proposed classification and coding scheme, a business component is described by identifiers (structured information such as name, industry type), followed by descriptor facets (unstructured information, such as rules, functionality). In their knowledge-based repository design, a database is used as the repository because it is efficient and effective

for storage, search and retrieval; eXtensible Markup Language (XML) is suitable to extensible numbers of descriptor facets.

Girardi and Ibrahim's solution for retrieving software artifacts is based on natural language processing. Both user queries and software component descriptions are expressed in natural language. Natural language processing at the lexical, syntactic and semantic levels is performed on software descriptions to automatically extract both verbal and nominal phrases to create a frame-based indexing unit for software components. But user queries and component descriptions are semantically formalized into the internal representation, canonical forms. Then the matching in between is reduced to compute the closeness of the query and the software component description, which is the distance of the two canonical forms. The retrieval system looks for closeness. A public domain lexicon is used to get the lexical information for both the query classification and the software classifications. To avoid rigidity, this solution employs partial canonical forms to allow some ambiguities and to infer all possible interpretations of a sentence.

Mingyang et al. proposed a component retrieval model combining knowledge intensive case-based reasoning technologies and conversational case-based reasoning methods. Case-based reasoning is a problem solving method. The main idea underlying this scheme is that when confronted with a new problem , similar or approximate problem solved in the past is analyzed which provides a break-through in solving the present problem. This scheme is divided into four phases: retrieve, reuse, revise and retain. This scheme focuses on retrieve phase. In retrieve phase, a new case (new problem description) is compared to the stored cases, and most similar one (ones) will be retrieved. Partial matching is adopted in the retrieve phase. Partial matching here refers to the matching of a group of features in order to return a best match, and where each feature typically has its own weight, distinguishes this technology from information retrieval and database access methods in general. However some such methods are knowledge-poor, which only consider superficial or syntactical similarities between a new case and stored cases, while other systems take both the syntactic similarity and semantic similarity into account by combining case-specific knowledge and general domain knowledge. The latter approach is referred as knowledge-intensive case based reasoning. Conversational case-based reasoning (CCBR) is an interactive form of case-based reasoning. It uses a mixed initiative dialog to guide users to facilitate the case retrieval process through a question-answer sequence. In the traditional CBR process, users are expected to provide a well-defined problem description (a new case), and based on such a description, the CBR system can find the most appropriate case. But usually users can not define their problem clearly and accurately. So instead of letting users guess how to describe the problem, CCBR calculates the most discriminative questions automatically and incrementally, and display them to users to extract information to facilitate the retrieval process.

Sugumaran and Storey[4] present a semantic-based solution to component retrieval. The approach employs domain ontology to provide semantics in refining user queries expressed in natural language and in matching between a user query and components in a reusable repository. The approach includes a natural-language interface, a domain model, and a reusable repository. The three major steps in the approach are: 1) initial query generation, 2) query refinement, and 3) component retrieval and feedback. Initial query generation is heuristic-based natural language processing to abstract keywords from user-

queries. Their approach for natural language processing of user query and component descriptions is based on the ROSA system [P10-10]. Query refinement is achieved by mapping initial query against ontology to ensure correct terms are used in the query. Ontology refers to a data-structure for storing the information in taxonomical way. In this case, ontology provides a mechanism to represent and store domain specific knowledge, which can be used to find the most related artifacts. In the component retrieval and feedback step, a closeness measure proposed by Girardi and Ibrahim[1] is employed to identify which are the most relevant components to the user's query.

3. PROPOSED SYSTEM

The proposed system 'Classification Of Software reusable components' is required to implement a classification scheme to build a library and provide an interface for browsing and retrieving components. Of these the main requirement is to develop a classification scheme which is used for classifying components. The system should support three operations viz. Uploading components, downloading components and search for components.

Uploading a component means storing the component in the repository. Developer should be able to upload his components into the repository. While uploading the components the system should make sure that components are not duplicated. If there are two components with similar properties they should differ at least in the language, developer or version and so on.

Downloading a component means the user can download the component on his system for reuse. The user should be able to see the file size and depending on his availability of the memory he can download the component.

Searching for components should be implemented such that the user can get the relevant components by entering a search query. However the system under development does not deal with natural language queries. The system should support an advanced search facility which provides the user to narrow down his search results to most appropriate components.

A helpful interface should be developed for this system, which should not lead to any ambiguity. The interface should be very easy to operate and provide good interaction between the system and the user.

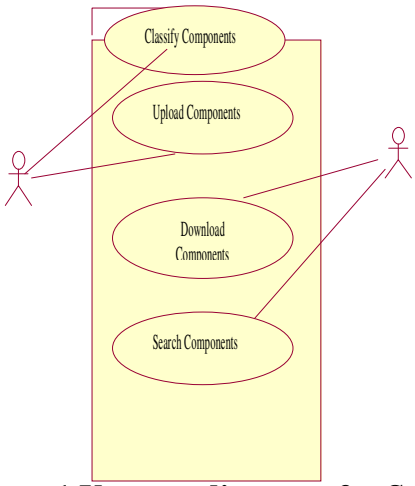


Figure 1 Use case diagrams for Software Reuse Repository

The proposed system can be depicted as shown in figure 2 as explained in the functional requirements; the system (big rectangle) provides the required functionality for uploading, downloading and searching components. The system also maintains the repository of the components. The system provides interface to the user's viz, administrator and users.

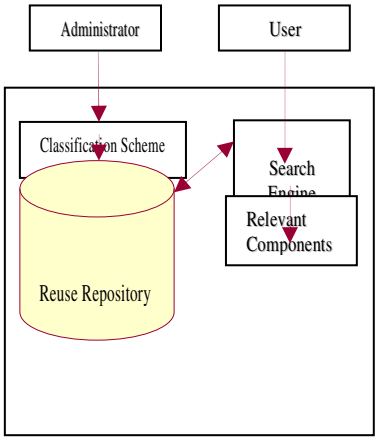


Figure 2 Proposed system model

3.1 Classification scheme

The intent of developing this classification is to integrate existing schemes. I used both attribute-value and faceted classifications together to achieve the classification. I have used the following attributes to identify the components:

- Component name
- Domain
- Language

Along with these attributes the following facets are used for classification of the components.

- File size
- version
- Time taken

The attributes when used in query can narrow down the search space to be used while retrieval. The attributes and facets listed here are for gathered from existing researches. To be more effective the system should be deployed and thoroughly used, then depending on the feedback system the system can be enhanced.

Here we used attributes and facets. The system also stores the descriptions of each component uploaded in the repository. If system stores most of the component's properties the system can serve better and can be used in different ways.

3.2 Algorithms

3.2.1 Uploading Components:

Input: component and attributes
Output: status of component
Step 1: capture user's input for component
Step 2: check for attribute similarities in existing components

Step 3: if no matching components found classify the component and store it in the repository

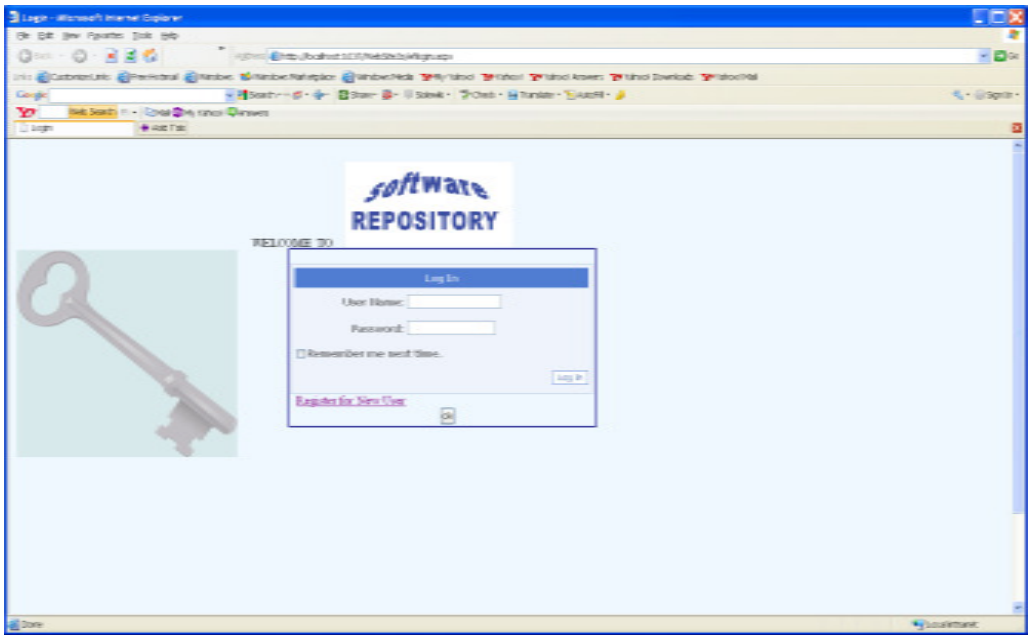
3.2.2 Searching Components

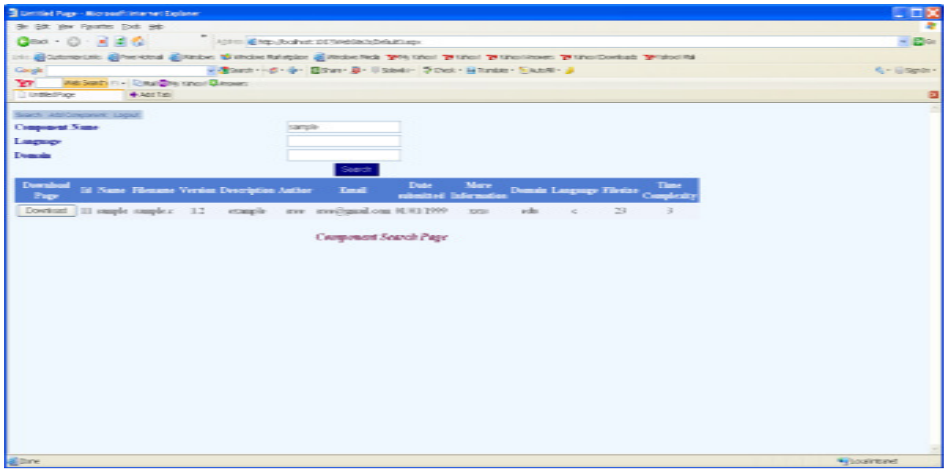
Input: Search query and some attributes
Output: list of components
Step 1: with user given criteria form a query that can be executed on the catalog
Step 2: execute the query to get the list of components that satisfy the attributes given by the user.
Step 3: from the list of components find the components that will match to the search query. The search query will be matched with component name, algorithm and function.
Step 4: all the components that are candidates for user requirements are presented to the user.

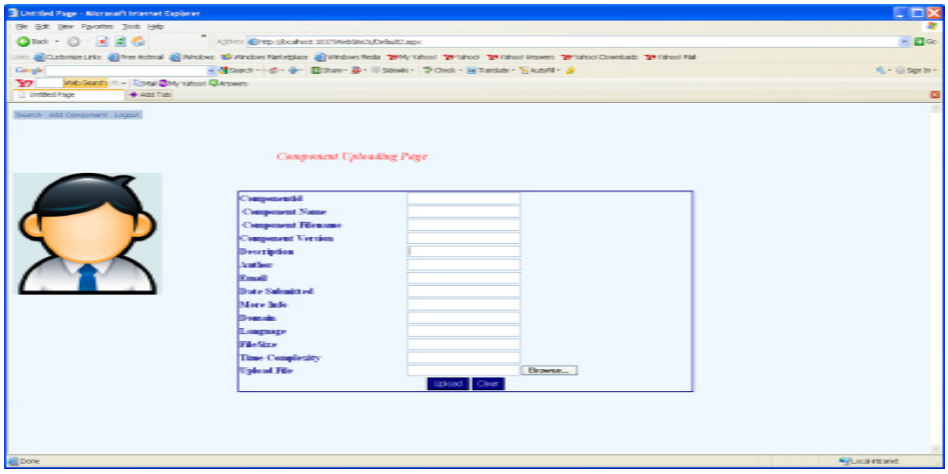
3.2.3 Downloading Components:

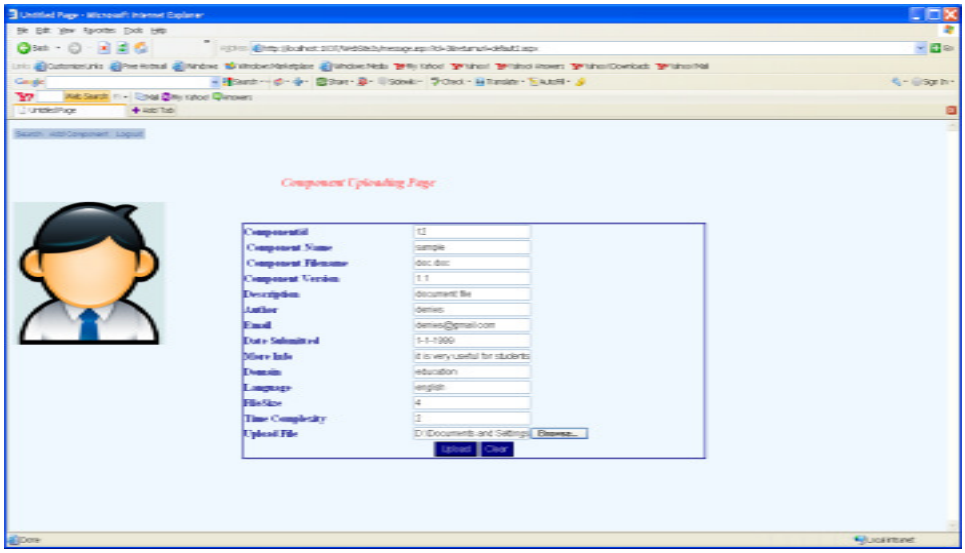
Input: Choose the component
Output: Component gets downloaded
Step 1: With user given criteria forms a query that can be executed on the catalog.
Step 2: Execute the query to get the component catalog downloaded
Step 3: Choose an option to save or view the file
Step 4: Depending on user choice the component can be viewed /saved respectively

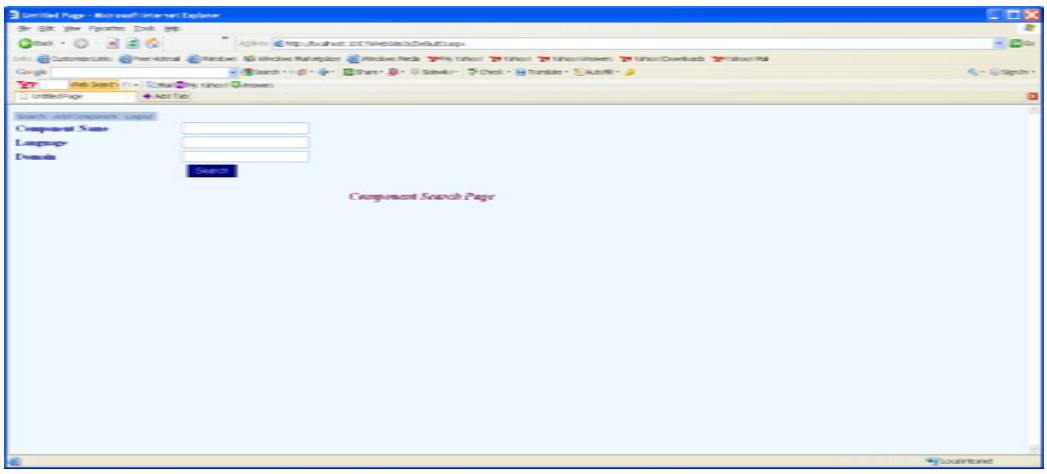
4. RESULTS

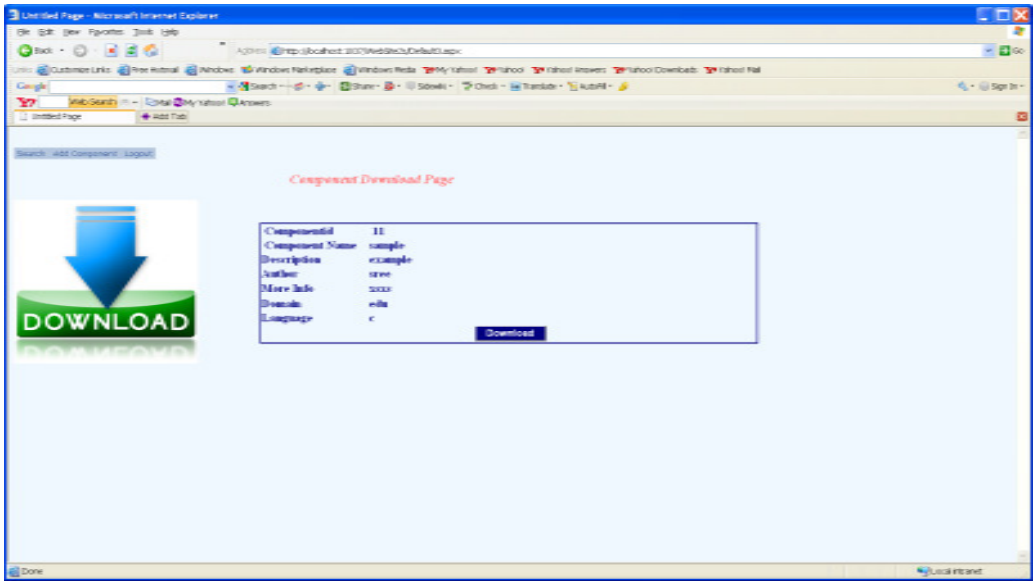



Component Search Page


Component Uploading Page


Component Uploading Page





5 CONCLUSION AND FUTURE SCOPE

There have been many attempts to classify reusable components using various techniques. The proposed classification system takes the advantage of the positive sides of each classification scheme, whilst hopefully rendering the negative sides redundant.
This classification scheme uses the attribute value and faceted schemes for different parts of a component.
Future work involved with this classification scheme will be to refine the scheme and formalize it for implementation. Though some classification scheme is developed it was not done with extensive study of all existing components. In order to be effective the scheme should be enhanced to meet the interests of people.

REFERENCES

[1] 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.

[2] Lars Sivert Sorumgard, Guttorm Sindre and Frode Stokke, "Experiences from Application of a Faceted Classification Scheme", 1993 IEEE,pp 116-124.
[3] William B.frakes and kyo kang, "software reuse research : status and future", IEEE transactions on Software Engineering, VOL.31 , NO . 7, JULY 2005.
[4] R.Prieto-Diaz and P.freeman, "Classifying software for Reuse",
IEEE Software, 1987, Vol. 4, No .1, pp. 6-16.
[5] Lars Sivert Sorumgard Guttorm Sindre and Frode Stokke, "Experiences from Application of a Faceted Classification Scheme" © 1993 IEEE, pp 116-124.
[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] Vicente Ferreira de Lucena Jr., "Facet-Based Classification Scheme for Industrial Automation Software Components"
[8] Ruben Prieto-Diaz, "Implementing Faceted Classification for Software Reuse" © 1990 IEEE, pp.300-304
[9] Klement J. Fellner and Klaus Turowski, "Classification Framework for Business Components", Proceedings of the 33rd Hawaii International conference on system Sciences- 2000, 0-7695-0493-0/00 © 2000 IEEE
[10] Vitharana, Fatemeh, Jain, "Knowledge based repository scheme for storing and retrieving business components: a theoretical design and an empirical analysis", IEEE Transactions on Software Engineering, vol 29, no. 7, pp, 649-664.
[11] William B.Frakes and Kyo Knag, "Software Reuse Research: Status and Future", IEEE transactions on Software Engineering, VOL.31 NO.7, JULY 2005
[12] R.Prieto-Diaz and P.Freeman, "Classifying Software for Reuse", IEEE Software, 1987, Vol.4, No.1, pp.6-16.
[13] Rym Mili, Ali Mili, and Roland T.Mittermeir, "Storing and Retrieving Software Components a Refinement Based System", IEEE Transactions of Software Engineering, 1997, Vol.23, No.7, pp. 445-460
[14] Hafedh Mili, Estelle Ah-Ki, Robert Godin, and Hamid Mcheick, "Another nail to the coffin of faceted controlled vocabulary component classification and retrieval", Proceedings of the 1997 symposium on software reusability (SSR'97), May 1997, Boston USA, pp.89-98.
[15] Hafedh Mili, Fatma Mili, and Ali Mili, "Reusing Software: Issues and Research Directions", IEEE Transactions on Software Engineering, Vol.21 No.6, June 1995.
[16] Gerald Jones and Ruben Prieto-Diaz, "Building and Managing Software Libraries", © 1998 IEEE, pp.228-236.
[17] Roger S.Pressman, "Software Engineering A Practitioner's approach", 5th Edition,Mc-Graw Hill.
[18] Ostertag, Hendler, Prieto-Diaz, Christine, "Computing Similarity in a Reuse Library System: An AI based Approach", ACM Transaction on Software Engineering and Methodology, vol. 1, no. 3, pp. 205-228.