REUSING COMPONENTS
Using
GENETIC ALGORITHMS

A Thesis

Submitted in partial fulfillment of the
requirement for the award of degree
Of

Master of Engineering
In
Software Engineering

Under the Supervision of
Mr. Rajesh Kumar Bhatia
Assistant Professor
Computer Science and Engineering Department
Thapar Institute Of Engineering and Technology, Patiala

Submitted By
Ms. Navjoti Pandhi
(8023111)

Computer Science & Engineering Department
Thapar Institute Of Engineering & Technology
(Deemed University), Patiala-147004 (India)

May 2004

Abstract

The need to improve software productivity and software quality has put forward the research on software reuse technology. Today's, complex, high quality computer based systems must be built within budget and time schedule. This demand leads to reuse of existing components. Software Reuse involves four steps definition, retrieval, adaptation and incorporation. A structured software library, where software components are properly classified is required so that the user can easily retrieve the software components. The main technical problems, which limit the practice of reuse, are effective structuring of the software repository and a mechanism to effectively retrieve from the repository. Structuring library of software components represented by natural languages may hinder the retrieval process due to the problem of ambiguity, incompleteness and inconsistency inherent to natural language. Formal Specifications, representing the software components are unambiguous, complete, and consistent and can be verified. The approach presented in this thesis uses Genetic Algorithms with Formal Specifications for clustering the software components, in which the number of clusters formed depends upon the similarity in cluster and similarity between clusters. The aim is to find optimized clusters in which components are classified. For cluster based retrieval the main aim is to find the cluster containing components similar to the given query.

DECLARATION

I hereby certify that the work which is being presented in the thesis entitled, "Reusing Components Using Genetic Algorithms", in partial fulfillment of the requirements for the award of degree of Master of Engineering in Software Engineering submitted in Computer Science and Engineering Department of Thapar Institute of Engineering and Technology (Deemed University), Patiala, is an authentic record of my own work carried out under the supervision of Mr. Rajesh K. Bhatia.

The matter presented in this thesis has not been submitted by me for the award of any other degree of this or any other University.

Ms. Navjoti Pandhi

This is to certify that the above statement made by the candidate is correct and true to best of my knowledge.

Mr. Rajesh Kumar Bhatia
Assistant. Professor
Computer Science and Engineering Department
Thapar Institute Of Engineering And Technology, Patiala

Countersigned by

Ms. Seema Bawa

H.O.D

Computer Sc. and Engg. Department

Thapar Institute of Engg. and Technology

Patiala.

Dr. D.S Bawa

Dean of Academic Affairs,
Thapar Institute of Engg. and Technology
Patiala.

The M.E. (Thesis) Viva-Voice examination of Navjoti Pandhi, Roll No. 8023111, M.E. (Software Engineering), Thapar Institute of Engineering and Technology, Patiala has been held on ……..

Supervisor **External Examiner**

Acknowledgement

A journey is easier when traveled together. Interdependence is certainly more valuable than independence. This thesis is the result of work carried out during the final year of my course whereby I have been accompanied and supported by many people. It is a pleasant aspect that I have now the opportunity to express my gratitude for all of them.

No amount of words can adequately express the debt, I owe to Mr. Rajesh Kumar Bhatia, Assistant Professor, Computer Science & Engineering Department, for his kind support, motivation and inspiration that triggered me for the thesis work. I owe him lots of gratitude for having me shown this way of research.

I wish to express my gratitude to Ms. Seema Bawa, Assistant Professor & Head, Computer Science & Engineering Department for her excellent guidance and encouragement right from beginning of this course. I am also thankful to all the faculty and staff members of the Computer Sc. & Engg. Department for providing me all the facilities required for the completion of this work.

No thesis could be written without being influenced by the thoughts of others. I would like to thank my classmates who were always there at the hour of the need and provided with all the help and support, which I needed. I am grateful to my brother Mr. Rohtash B. Pandhi who had the patience and hard work to study the core Software Reuse and helped me with his kind suggestions.

At last but not the least I would like to thank "**The Creator of Destinies**" for not letting me down at the time of crises and showing me the silver lining in the dark clouds.

Navjoti Pandhi

Roll no. 8023111

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Number Page

Organization of Thesis

In chapter 1, software component retrieval for reuse has been explored. Reuse artifacts, artifacts characteristics and reuse benefits have been dealt with.

Chapter 2 deals with the various classification and retrieval techniques. Reuse effectiveness in terms of recall and precision has also been discussed.

Chapter 3 Component Specifications, Similarity measure is discussed in this chapter. Proposed Model based on formal specifications and genetic algorithms has been discussed

The work done has been concluded along with future directions in chapter 4

Chapter 1

Software Reuse

The operation of the modern world depends upon software systems. It is like the bone marrow of body of modern world. Each and every field of modern world depends on it. So do the banking system, the electrical utility grid and telecommunication. Competition between software companies is becoming both more intense and more widespread. More and more companies depend on mission critical business information systems to manage customer interaction, speed goods to their destination, and manage finances in an ever more complex, global business environment. Often companies compete on small differences in quickly introduced, innovative services. Slowing advancements in software productivity and increased demands mean that most software organizations are not able to produce manageable, high quality, cost-effective software. Improving business performance often means improve their software development performance. In all cases, success means that products and services that rest upon a software base must get their software " faster, better, and cheaper".

Faster the software has to meet a market window. Competitive organizations set that time.

Better the software has to serve the requirements of the process it is to support and later, when serving its process, it has to do with few failures.

Cheaper the software has to be less expensive to produce and maintain.

There are two ways to achieve these goals. Either the software must be produced more efficiently or large portions must be reused. The first way is to increase the effectiveness of the organization that produces the software. The second way is reuse. Software reuse is seen as effective solution to build software faster, having high quality and cheaper. Few years ago when there was no software, one has to build software from scratch. The reusing of quality components ensures increase in productivity and overall quality. This may be so because the effort needed to create reusable software components, locate them, adapt them and integrate them in a specific application has been usually lesser than the effort needed to create the application from scratch. Still, there are many factors that limit the practice of software reuse, among them the lack of tools to construct reusable software components and the lack of tools for fast and effective retrieval of components. But still to improve the productivity and to improve the quality of software one has to use the existing software.

1.1. What is Software Reuse

Reuse is the use of existing concepts or objects in a new situation. It involves storing the encoding of existing concepts for future reference, matching the new situations with the old situations, duplication and their adaptation to suit new requirements.

Reusability is a measure of the ease with which one can use those previous concepts or objects in the new situations [8].

Reuse can be achieved through different modes. *Compositional* reuse involves constructing new software products by assembling existing reusable assets. G*enerative* reuse involves the use of application generators to build new applications from high-level descriptions

1.2. Motivations For Reuse

In today's competitive market scenario, the software companies has to shorten the time required to bring a product to the market, reducing software development and maintenance costs, and increasing the quality of their software. To meet their business goals organizations are turned to reuse strategy. Reuse helps in achieving some or all of the following goals [24]:

- Increase Productivity

  After an initial investment, reuse of existing assets will enable projects to decrease the cost of developing and maintaining software.

- Shorten Time-To-Market

  Reusing software shortens the critical path in delivering a product.

- Improve Software Quality

  Software that has been used multiple times has fewer defects than newly developed software.

- Provide Consistency and Interoperability Across Products

Standard interfaces and common use of components across products facilitates ease of use and interoperability. For example, when several software products reuse the same user interface scheme, terms and conventions are used consistently. Reuse also contributes to interoperability. For example, when a component that contains an error-message handling routine is reused by several systems, these systems can expect consistent behavior.

- Ensure Product/System Conformance with User Requirements Through Prototyping

  As the components are already exists it is easy to validate the user requirements. This prototyping will also enable detection and resolution of defects earlier in the software life cycle - avoiding more costly fixes later in the life cycle.

- Reduce Risk

  Risk in creating new software is reduced when available reusable components already encompass the desired functionality and have standard interfaces to facilitate integration.

- Leverage Technical Skills and Knowledge

  It enable specialists to optimize software architectures and assets which may then be reused by others who are focusing on meeting product features and market needs.

- Improve Functionality and/or Performance

Reuse allows improving functionality and/or performance of new developed software. The performance of the software is assured to be good as the components reused are already being tested and executed.

1.3. Reuse Drawback

In practice the gain in productivity and quality cannot be achieved due to some preconceptions of developers and management. The claims commonly put forward by programmers include [24]:

- Reusing code, as compared with development of entirely new systems, is boring.
- Locally developed code is better than that developed elsewhere.
- It is easier to rewrite complex programs from scratch rather than to maintain it.
- There are no tools to assist programmers in finding reusable artifacts.
- In majority of cases, developed programs are too specialized for reuse.
- Adopted software development methodology does not support software reuse.
- Reuse is often ad-hoc and is unplanned.
- There is no formal training in reusing code and designs effectively.
- Useful reusable artifacts are not supported on the preferred development platform.
- The reuse process is too slow.
- Interfaces of reusable artifacts are too awkward to use.
- Code with reusable components is often too big or too inefficient.

- Programs built of reusable components are not readily transportable.
- Reusable components do not conform to adopted standards.
- Reuse techniques do not scale up to large software projects.
- There are no incentives to reuse software.

1.4. Reusable Artifacts

The objects that can be reused are known as reusability artifacts, this includes any of the following software components [24]:

- Project Plans: The basic structure and much of the content of a software project plan can be reused from project to project. This reduces time to develop plan and uncertainty associated with establishing schedules, risk analysis and other features.
- Cost Estimate: Often different projects have similar function; it may be possible to reuse estimates for that function with little or no modification.
- Architecture: There are relatively little distinct programs and data architecture, even when different application domains are considered.
- Designs: Architectural, data, interface, and procedural designs developed using conventional methods are candidate for reuse. More commonly, system and object designs are reused.
- Source Code: Verified program components written in compatible programming language is candidate for reuse.
- Human Interfaces: Probably the most widely reused software artifact is Graphical User Interface, which is commonly reused.

- Data: Among the most commonly reused artifacts, data encompasses internal tables, lists, and record structures as well as files and complete databases.
- Test cases: Whenever a design or code component is to reuse, the relevant test cases are also reused.

These artifacts constitute a reuse library. These libraries must contain not only reusable components but are also expected to provide certain types of services like storage, searching, inspecting and retrieval of artifacts from different application domains, and of varying granularity and abstraction, loading, linking and invoking of stored artifacts, specifying artifact relationships, etc. The major problems in software libraries are classification and retrieval from these libraries.

1.5. Artifact Characteristics

Certain software artifacts have been reused these artifacts usually share a number of characteristics, to increase reusability [24],

- Expressive: They are of general utility and of adequate level of abstraction, so that they could be used in many different contexts, and are applicable to variety of problem areas.

- Definite: The artifacts should be constructed and documented describing their purpose, their capabilities and limitations are easily identifiable, interfaces, required

resources, external dependencies and operational environments are specified, and all other requirements are explicit and well defined.

- Transferable: The artifacts should be easily transferable from one environment to different environment or domain.

- Additive: The artifacts should be easily fitted to new application without requiring enormous modifications or causing adverse side effects.

- Formal: Reusable artifacts should be described using formal and semiformal notations so that the verification of artifacts should be easy.

- Machine Represent able: Those of the artifacts which can be described in terms of computationally determined attribute values, which can easily be decomposed into machine represent able parts, which can be accessed, analyzed, manipulated and possibly modified by computer-based processes, have a clear potential for becoming part of a flexible reuse library; those artifacts can be easily searched for, retrieved, interpreted, altered and finally integrated into larger system.

- Self-contained: Reusable artifacts which embody a single idea are easier to understand, they have less dependencies on external factors, whether environmental

or implementation, they have interfaces which are simple to use, they are easier to extend, adapt and maintain.

- Language Independent: The artifacts should not be embedded with implementation details in reusable artifacts.

1.6. Classifying And Retrieving Components

Consider a large university library. Tens of thousands of books, periodicals, and other information resources are available for use. But to access these resources, a categorization scheme must be developed. To navigate this large volume of information, librarians have defined a classification scheme that includes a library of congress classification code, keywords, author names and other index entries . All enable the user to find the needed resource quickly and easily. Similarly in component library tens of thousands components are stored. How can user finds out the appropriate one. The solution to this problem is classifying the software libraries in a particular order and use effective retrieval technique. A brief introduction of classification and retrieval techniques is given.

1.6.1. Classification of Reusable Component

There are three fundamental techniques for classifying the software library

(1) *Enumerated classification:* is a well-known retrieval method. In this technique the information is placed in categories and structured in hierarchy of sub

categories. The main motive behind this scheme is that the information is divided into smaller pieces. The issues involved in using an enumerated classification include its inherent inflexibility and problems with understanding large hierarchies [16]. There is a tradeoff between the depth of a classification hierarchy and the number of category members. Some domains will lend themselves to many small classes. The effect is that users unfamiliar with its structure will become lost in the morass of possible classes. Other domains will have few categories, but must necessarily contain many members. In this case, selection of a class is only a first step in the retrieval process, as the user must then search a large number of category members for relevant information. Enumerated classification requires users to understand the structure and contents of the repository for effectively retrieve information behavior.

(2) *Attribute-value classification*: It uses the text from a document for indexing. These texts are extracted from documents attached to components. The common terms like ‘a’, ‘an’, ‘to’ and ‘the’ are excluded and terms which best represents the components are used as index for components. Users specify a query using key words that are applied to the indices to find matching documents [16].

(3) *Faceted Classification*: It avoids enumerating component definitions in a hierarchy by defining attribute classes that can be instantiated with different terms grouped into a fixed number of mutually exclusive facets. Users search for

components by specifying a term for each of the facets. Within each facet, classification techniques are used to help users choose appropriate terms. This is very similar to the attribute-value structures used in a number of frame-based retrieval techniques in artificial intelligence

1.6.2. Retrieval

Through the retrieval process, potential reusable software components, satisfying specific user's requirements, are searched and selected from software bases. The selected components either exactly satisfy the user requirement or partially stasify the user requirement. Browsing a hierarchy of software components in the software base, by following hypertext or hypermedia links in the software base or by linear retrieval can do the searching [19]. Linear retrieval is the typical activity of an information retrieval system, where the user describes the features of a required object through a query and the system retrieves a list of objects having some degree of similarity with these features.

1.7. A Reuse Life Cycle

Software reuse involves two major complementary activities.

(1) Developing reusable software, and

(2) Reusing existing software

The first activity, named Development FOR Reuse [13] involves the construction of reusable software components to be used in the development of similar applications in

a particular domain. The second activity, named Development WITH Reuse, concerns the construction of applications with specific requirements by using available reusable software components in the application domain from a software base.

Development FOR Reuse requires good knowledge of the selected application domain and experience in the development of similar applications in that domain. Development FOR Reuse is an evolutionary task that uses the accumulated experience acquired during Development WITH. The reuse life cycle is shown in Figure1.1[13].

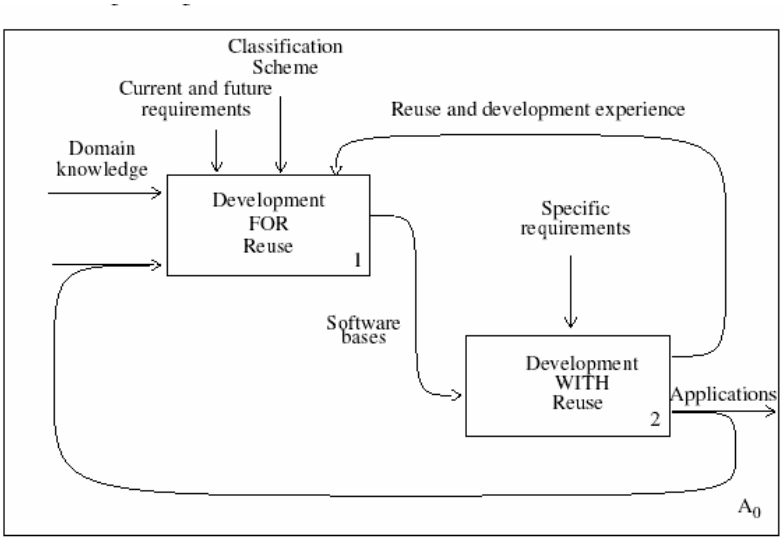


Figure 1.1 The Reuse Life Cycle

Enough techniques for finding and selecting components have received attention. Various classification and retrieval techniques are used to extract components from the repository. These techniques are discussed in detail in second chapter.

1.8. Summary

In this chapter the concept of software reuse, reusability, reusable artifacts and their characteristics has been explored. We have discussed the benefits of reusing software components. Next chapter deals with the various classification and retrieval techniques for software reuse.

Chapter 2

Classification and Retrieval Techniques

The major activities in any information retrieval systems are classifying and retrieving the components. Classification of information means specifically a logical organization of information, such as document representatives or organizing the software library so that the information retrieval becomes easy. The development in classification of components has been fairly recent. The reason behind the slowness of development in this area of information classification and retrieval, is that no one realized for a long time that computers would not give an acceptable retrieval time for very large repository size unless a particular classification scheme is not applied on it. Still the, owners of large databases are still inclined to try out new classification schemes promising faster and effective retrieval. The slowness to recognize and adopt new techniques is mainly due to the lack of the experimental evidence to support them. Retrieving the software components means extracting software components from the repository. There are number of techniques for retrieving the software components. In principle, information storage and retrieval is simple. Suppose there is a store of documents and a person asks a question to which the answer is a set of documents satisfying the information need expressed by his question. He can obtain the set by reading all the documents in the store, retaining the relevant documents and discarding all the others. In a sense, this is known as 'perfect' retrieval. A user either does not want to spend the time reading the entire document collection. He needs a organization of

the documents in a particular way so that he can easily find the documents satisfying his query.

2.1. Functions and Concepts of an Information Retrieval System

The main purpose of information retrieval system is to satisfy the query representing the need of user. A query or request representing the information retrieved is fired to the repository and the components fulfilling the request are retrieved. The major functions of an IRS [13]:

1. Indexing
2. Search strategy
3. Matching or similarity analysis
4. Evaluation of retrieved items and feedback for query refinement.

The process of indexing or cataloguing is classifying the documents in a particular way is done through he conceptual analysis of information item.

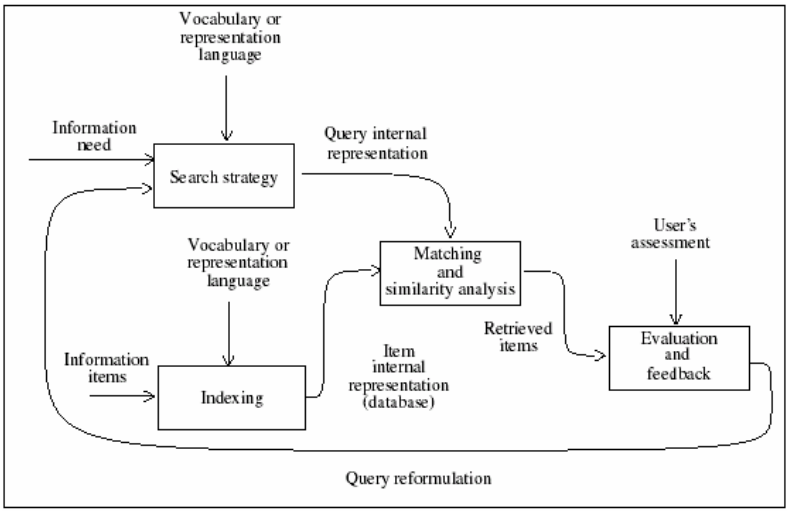


Figure 2.1 The Major Functions of Information Retrieval System

2.2. Effectiveness of Retrieval System

The choice of particular classification and retrieval scheme depends on their effectiveness. Much of the research and development in information retrieval is to improve the effectiveness and efficiency of retrieval and classification scheme. Efficiency is usually measured in terms of the computer resources used such as memory used and C.P.U. time. Efficiency measured in machine independent way is difficult. For evaluating an information retrieval system, main concern is to provide information to user so that he can make decision as to whether there is need of such a system and whether it will be worth. The evaluation of particular search strategy will lead to whether to adapt that strategy or not. The other problem is how one can measure that a particular strategy is useful for user. In fact, as early as 1966, Cleverdon listed six main measurable factors to measure the effectiveness [13]:

(1) The *coverage* of the repository, that is, the extent to which the system includes relevant matter;

(2) The *time lag*, that is, the average interval between the time the search request is made and the time an answer is given;

(3) The form of *presentation* of the output;

(4) The *effort* involved on the part of the user in obtaining answers to his search requests;

(5) The *recall* of the system, that is, the proportion of relevant material actually retrieved in answer to a search request;

(6) The *precision* of the system, that is, the proportion of retrieved material that is actually relevant.

The important factors for measuring the effectiveness of retrieval system are recall and precision. In other words it is a measure of the ability of the system to retrieve relevant documents while at the same time not retrieving non-relevant one. It is assumed that the more effective the system the more it will satisfy the user. It is also assumed that precision and recall are sufficient for the measurement of effectiveness. Precision is the ratio of the number of relevant documents retrieved to the total number of documents retrieved, and recall is the ratio of the number of relevant documents retrieved to the total number of relevant documents.

Table 2.1 Retrieval Effectiveness

	RELEVANT	NON-RELEVANT
RETRIEVED	$A \cap B$	$\overline{A} \cap B$
NOT RETRIEVED	$A \cap \overline{B}$	$\overline{A} \cap \overline{B}$

$$PRECISION = \frac{|A \cap B|}{|B|}$$

$$RECALL = \frac{|A \cap B|}{|A|}$$

2.3. Approaches to Software Classification and Retrieval

In this section various approaches to software component classification and retrieval from repository are discussed. It is essential to effectively classifying and retrieving the components from repository so that the effectiveness of reuse is higher.

- Classification schemes
- Automatic indexing
- Knowledge-based Approaches
- Formal Specifications
- Behavior based retrieval
- Browsing
- Hypertext
- Neural network based Approach
- Fuzzy logic based retrieval
- Genetic Algorithm based retrieval

2.3.1. Classification Schemes

A major problem in software reuse is organizing collections of reusable components for effective search and retrieval. Faceted classification offers certain features that not only improve search and retrieval, but also contribute to the development of a standard vocabulary for software attributes. Faceted classification is synthetic means components are grouped together according to predefined keywords from faceted lists. This approach provides higher accuracy and flexibility in classification. Faceted

classification was introduced by Ranganathan in the late 1930s and is widely used in libraries throughout Europe and India [6]. Classification in a faceted scheme is straightforward. A faceted scheme may have several facets and each facet may have several terms. A hypothetical faceted scheme with two facets, entities and activities, is introduced in Table 2.2 to illustrate the classification process [6].

Table 2.2 Faceted Schemes

{entities}	**{activities}**
Designs	Analysis
Programs	Design
Structures	Evaluation
Systems	Programming

To classify the title in the example, term is selected from each facet which best describes each of the concepts in the title. "Systems" is selected from the entities facet, and "programming," from the activities facet. The example in Table 2.3 [6] is a faceted scheme developed for Unix components. It consists of four facets: the function performed by the component, the objects manipulated by the function, the data structure where the function takes place, and the system to which the function belongs. The example shows the class of Unix components that locate line numbers in a file and are part of a line editor.

Table 2.3 A Simplified Faceted Scheme For Unix Commands

DOMAIN → Unix Tools			
{by action}	{by object}	{by data structure}	{by system}
get	file-names	buffer	line-editor
put	identifiers	tree	text-formatter
update	line-numbers	table	
append	character	file	
check	number	archive	
detect	expression		
locate	entry		
search	declaration		
evaluate	line		
compare	pattern		
make			
build			
start			

A classification scheme for collections of reusable software components should meet the following criteria:

1. It must accommodate continually expanding collections, a characteristic of most software organizations.
2. It must support finding components that are similar, not just exact matches.
3. It must support finding functionally equivalent components across domains.
4. It must be very precise and have high descriptive power (both are necessary conditions for classifying and cataloging software).
5. It must be easy to maintain, that is, add, delete, and update the class structure and vocabulary without need to reclassify.

6. It must be easily usable by both the librarian and end user.
7. It must be amenable to automation.

Faceted classification meets most of these demands. A conceptual graph structure was superimposed on each set of facet terms to measure class similarity.

2.3.2. Automatic Indexing

Before a computerized information retrieval system can actually operate to retrieve some information, that information must have already been stored inside the computer in some order. The computer, however, is not likely to have stored the complete text of each document in the natural language in which it was written. It will have, instead, a document representing the repository, which may have been produced from the documents either manually or automatically. To produce such document the starting point is text analysis. Text analysis process is analyzing the complete document text, an abstract, the title only, or perhaps a list of words only. From it the process must produce a document representative in a form to which the computer can handle.

An index language is the language used to describe documents and requests. The elements of the index language are index terms, which may be derived from the text of the document to be described, or may be arrived at independently. Index languages may be described as pre-coordinate or post-coordinate, the first indicates that terms are coordinated at the time of indexing and the latter at the time of searching. In pre-coordinate indexing a logical combination of any index terms may be used as a label to identify a class of documents, whereas in post-coordinate indexing the same class

would be identified at search time by combining the classes of documents labeled with the individual index terms.

The vocabulary of an index language may be controlled or uncontrolled. The former refers to a list of approved index terms that an indexer may use. The controls on the language may also include hierarchic relationships between the index terms. In fully automatic retrieval system, the keywords and phrases are automatically extracted from the documents attached. Automatic indexing methods are of two types [13]:

- Statistical methods
- Linguistic methods.

2.3.2.1. Statistical approaches

Statistical approaches to information retrieval are based on that the keywords or phrases with high frequency are not useful in distinguishing or discriminating the stored components. Statistical approaches based on single words in indexing and searching strategies, are ambiguous in meaning, because single words are not expressive enough to describe the components. Therefore, the phrases are used to represent the components. Syntactic approaches are suggested in order to reduce the number of wrong phrases [13]. The requirement of statically approaches is that the information items like documents attached with, are able enough to represent the components based on frequency of terms, which is not usually the case of documents describing the functionality of software components.

2.3.2.2. Linguistic approaches

Some form of natural language analysis should improve retrieval effectiveness and other processing operations. With the knowledge acquisition techniques helps in construction of knowledge bases for natural language understanding. Linguistic information is extracted from documents attached without full understanding of their meaning. Document retrieval is different from other tasks requiring Natural Language Processing like story understanding, because it is not necessary to make a complete unambiguous interpretation of a text to ensure reasonable retrieval effectiveness.

Most linguistic approaches to information retrieval, Natural Language Processing techniques have been applied at the morphological, lexical, syntactic and semantic level.

- **Morpholexical level**

In morpholexical processing techniques looking up a dictionary identifies both the standard forms and grammatical classes. For example, lexical processing would determine that the word adding is either the present continuous tense of the verb add or and adjective. The ambiguity in the indexes is removed at the syntactic level. In this analysis the words and their standard forms are identified according to their grammatical category. For instance, for the word searching, both the standard forms

search and searching will be identified, associated with the verb or adjective grammatical categories, respectively. Dictionaries are used to represent the index terms of components and queries by the word sense not only the words themselves. These approaches would retrieve the documents with words having multiple meanings. The dictionary provides a definition for each sense of a word and retrieves the items in context to the words in the queries.

- **Syntactic level**

In syntactic analysis, the structure of the sentence is determined. The syntax is specified in a grammar having the set of rules that determine which ordering of words are allowed and some parsing strategies are used to recognize the structure of the sentence. It have been mostly restricted to identify noun phrases and therefore simple than the other techniques required the natural language processing. Syntactic analysis permits the identification of different syntactic compounds and the phrases can be extracted from each syntactic compound. Thus, some wrong phrases like the ones obtained by statistical methods are avoided.

- **Semantic level**

The semantic level of NLP deals with meaning of the sentence and maps grammatically parsed text into a knowledge representation formalism. It should disambiguate the meaning of sentences. Sentences can be

ambiguous either because words within them are ambiguous or because they have many syntactic interpretations. Some approaches for semantic processing in information retrieval are based on the definition of a formal language into which input text could be turned. Other approaches use knowledge representation frameworks from artificial intelligence, like semantic networks, frames or Schank's primitive acts to represent knowledge by specifying primitives or simple concepts and then combining them to define more complex concepts [13].

Full understanding of documents on unrestricted domains is, unfortunately, not feasible with current NLP technology and successful NLP depends on putting limits on the need for real world knowledge and human experience. It also depends on exact knowledge of how human language works.

2.3.3. Knowledge Representation Approaches

The text automatic classification method is based on the content analysis automatically to allocate the text into pre-determined way. The methods of text automatic classification mainly used in information retrieval techniques. Traditional information retrieval techniques mainly retrieve relevant documents by using keyword-based or statistic-based techniques. One central step in automatic text classification is to identify the major topics of the texts. Knowledge-based approach is simple for classifying the

text automatically. In this type of system some kind of lexical, syntactic and semantic analysis is done without completely understanding of documents. A knowledge base is formed which store the semantic information about application domain and about natural language itself. This approach is powerful than most of the traditional approaches. The approach uses topic identification algorithm the principal process for the Knowledge based text classification is illustrated as in Fig 2.2[26]. The semantic representation structure used in this is case frame. These are described in most NLP and AI literature.

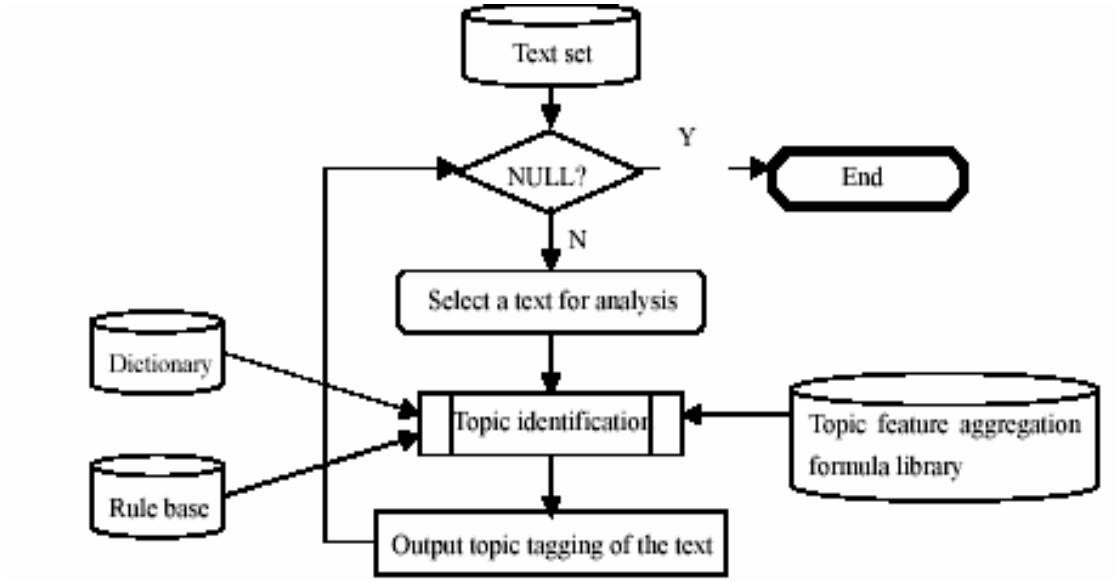


Fig 2.2 Knowledge Based Text Classification

2.3.4. Formal Specifications

Formal specifications are the mathematical notations to represent the properties of an information system have without considering the way in which these properties are achieved. They describe what the system can do without describing how it is done [23].

2.3.4.1. Specification Matching

Specification matching is a process of determining if two software components are related by matching the specifications describing the components. The components are classified using formal specification language. In retrieval process the search is for retrieving the components fulfilling the required behavior i.e. those components are retrieved which matches the given query. In retrieval, a close match is wanted; as in other information retrieval techniques. The search is not just for exact match between components, but many relaxed match are also considered. The signatures of components describe a component's type information; specifications describe the component's dynamic behavior. Specifications more precisely characterize the semantics of a component, rather than just its signature. Let the two components

C 1=<C_{sig} , C_{spec} >and C2=<C_{sig} , C_{spec} >

C_{sig} = Signature Specifications

C_{spec} = Formal Specification

Then the specification matching between the components is defined as[10]:

Match<C1, C2>= match<$C1_{isig}$, $C2_{spec}$> and match<$C1_{spec}$, $C2_{9spe}$>

Two components C1 and C2 match if (1) their signatures are matched; (2) their specifications are matched partially or fully.

2.3.5. Behavioral Retrieval

While reusing the software components the components behavior is one of the most crucial factor for building the system. In predefined retrieval methods approximate behavioral retrieval by using surrogate representations to describe behavioral properties of components. In behavioral retrieval the executability of software components are exploited. Programs are executed using components, and the responses of components are recorded. Retrieval is achieved by selecting those components whose responses are closest to a pre-determined set of desired responses.

The behavioral retrieval scheme is outlined as follows:

- An abstract view of component is taken in terms of their responses to programs.
- These collections of responses are meaningfully partitioned.
- A lattice by the partial order can be used to find best behavioral approximations to the component desired

2.3.5.1. Model for Execution

An execution model describes how components execute programs and generate output responses [5]. A component is represented as a relation between programs and responses. This is because in general, a program execution can yield several responses and a response may be evoked by more than one program. Formally, a component *c* can be declared as:

$$c : Prog \leftrightarrow Response$$

A program p ∈ Prog is modeled as a sequence of calls on the component's interface. A response is a sequence of values in correspondence with a program. The execution of program as follows when a program is modeled to a component, a component will respond by executing each call in the program and producing a corresponding output sequence. These responses are collected.

These collections of responses are meaningfully partitioned. A meaningful partial order over behaviors is defined. A lattice by the partial order can be used to find best behavioral approximations to the component desired

2.3.6. Browsing

Navigation or browsing is effective only for small systems. For large databases, information retrieval through queries becomes crucial. Conklin had suggested that search and query mechanisms could present information at a manageable level of complexity and detail. Foe effectively accessing information stored in a hypermedia network requires the query-based access to reach the desired goal. The browsing technique is applied to retrieve when the user is not sure for his goal. Links are considered as indexes, providing immediate access to required information without navigation through the document space. While node can be considered part of the document space, indexes or index nodes can be treated as part of the "index space". In traditional full-text document retrieval systems, the document space and index space are together. In hypertext systems a hierarchical index space and a networked document space is used for information retrieval. Usually the belief networks and

Bayesian networks are used. Belief networks or Bayesian inference networks are directed, acyclic dependency graphs where nodes represent components and links or edges represent the similarity between the components. User's likes and dislikes are transmitted recursively from all nodes representing components to nodes in the index space. Based on this, appropriate values are assigned to index space nodes. If a component represented by a node p has similarity with component represented by node q, a directed graph is drawn from p to q [7]. The values of the index of nodes show the interest of user. A hierarchical index space greatly enhances the information retrieval process in hypertext or networked document spaces. The retrieval effectiveness depends upon the ranking of nodes according to their probability of satisfying the query. Based on a natural language query, the system will perform a search on the network to find the most appropriate node. Matching the various terms in the query and evaluating the matched expressions by applying similarity functions do the node identification.

2.3.7. Hypertext

The conventional information retrieval techniques focus on "what to where" (we know what we want, but we wish to find out where in the database it is), hypertext browsers focus on "where to what" (we know where we are, but we want to know what is there) [26]. A hypertext system represents a non-linear form of documents. In hypertext retrieval technique the user is allowed to discover retrieval cues that can be used for query formulation. In hypertext techniques nodes represents various kind of

information. Links represents the relationship between the nodes [26]. Query facilities can supplement hypertext browsing by providing the user with a set of relevant nodes for browsing. User query produces a list of references to items that can be units, indexes or guided tours. These can be used as dynamic access structure from which further navigation can originate. The navigation space resulting from such a query is called a hyper view. All subsequent requests will be interpreted only within this restricted space.

2.3.8. Neural Network Based Retrieval

Neural network are used to construct the software library. The library is semantically organized i.e. according to functional similarity of the stored component. The components with similar behavior should be stored near to each other.

Self-organizing maps are unsupervised neural network are used to organize component library. The input data represents the feature of the storing component. This input data is mapped to a grid of neurons [1].

The output unit assigned the weight vector of n dimensional. During the learning process the input vectors are repeatedly presented to all output units. The outputs produced by these components are measured in terms of their similarity.

Then the adaptation of the SOM will take place. The weights are changed accordingly [3].

$w_j(t+1)=w_j(t)+\eta(t)*\xi_{ij}(t)\times[x(t)-w_j(t)]$

w_j (t+1) is the new weight assigned to output unit i. η and ξ_{ij} , the two scalar functions are used for grading the adaptation. The first function η, is a gain function which produces a so called learning rate in the range of [0, 1] as its output. The second function, ξ_{ij} , is a neighborhood function. The weight vectors are adapted in such a way that they get nearer to the current input vector. Figure 2.3 contains a schematic representation of the learning process [3].

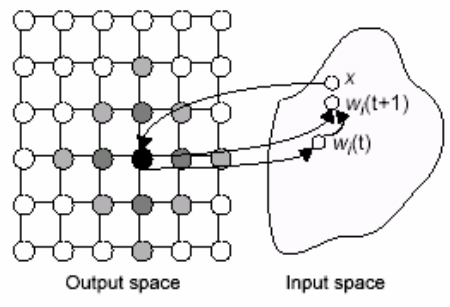


Fig 2.3 Grid of Neurons

For retrieving the component the required functionality is described in a query. This query in turn can easily be transformed into a feature vector, which is fired on to the grid, and the best-matching stored components are searched for. The best matching unit and its neighbor's components are retrieved.

2.3.9. Genetic Algorithms Based retrieval

In this technique components are specified with faceted classification. Components, with similar properties, are classified into clusters. Genetic Algorithms are used to form clusters with similar components so that user is able to find components easily. Genetic Algorithms are applied to clustering so that reusable components are classified

into optimized clusters $P^k = \{C_1, C2...C_k\}$ whose number is not fixed. The components are represented by a chrosome and its corresponding cluster is represented by gene as shown in Figure 2.4 [21].

Component	1	2	3	4	5	6	7	8	9	10
Cluster	2	3	3	1	5	3	1	6	1	2

Fig 2.4 Chromosome Representing Components and Clusters

The cluster 1 has three components 4,7 and 9. There are two objective functions one for clustering and another for retrieval. For clustering the objective function is based on the concept that similarity within the cluster is high and similarity between the clusters is low. The goal of multi-way clustering is to find clusters $P^k = \{C_1,C_2,.....,C_k\}$ minimizing the objective function.

For retrieving the objective function is based on the concept of retrieving those clusters, which has high precision and high recall. For retrieving the components user gives the queries these queries are matched with the cluster. The cluster with maximum value for the objective function is retrieved.

2.3.10. Fuzzy logic based retrieval:

The software components are described using the descriptors in fuzzy based retrieval techniques describing the key characteristic of components. These descriptors are composed from term pairs, reflecting the functionality of

component and are constructed from the code and accompanied documentation [15]. These features are list are assigned a relevance measure expressed by a weight, a number between 0 and 1.

The classification of software components is done using a descriptor in fuzzy based retrieval techniques. Software descriptor is a list of features describing the key characteristics of a software component. These features are extracted automatically. The weights assigned to features are viewed as the fuzzy sets and, fuzzy techniques are used for retrieval.

A feature weighing function is used to assign the weights to descriptors. Fuzzy matching between the software descriptors is used for computing the conceptual distance between the components. Compatibility between two descriptors are given by a fuzzy relation. The relation gives the value between 0 to 1, which describes the similarity between the two components. A pre set threshold value is compared with compatibility value, if the value is greater than the threshold value the component is retrieved. The similarity value depends upon number and weight of the common features and their importance.

2.4. Summary

In this chapter problem related with classification and retrieval of software components is discussed. Reuse effectiveness in terms of recall and precision has been discussed. Various retrieval and classification techniques have been explored. Next chapter deals with the classification and retrieval of software components with Genetic Algorithms

Chapter 3

GA Based Proposed Model

The genetic algorithms are used to solve NP hard problems. The genetic algorithms transform a population of individual objects, each with an associated fitness value, into a new generation of the population using the Darwinian principle of reproduction and survival of the fittest and naturally occurring genetic operations such as crossover and mutation. Each individual in the population represents a possible solution to a given problem. The genetic algorithms are used to find a very good or best solution to the problem [28].

Clustering of software components in a large repository is a NP hard problem. As the number of components increases the ways to classify these components into clusters, increases exponentially.

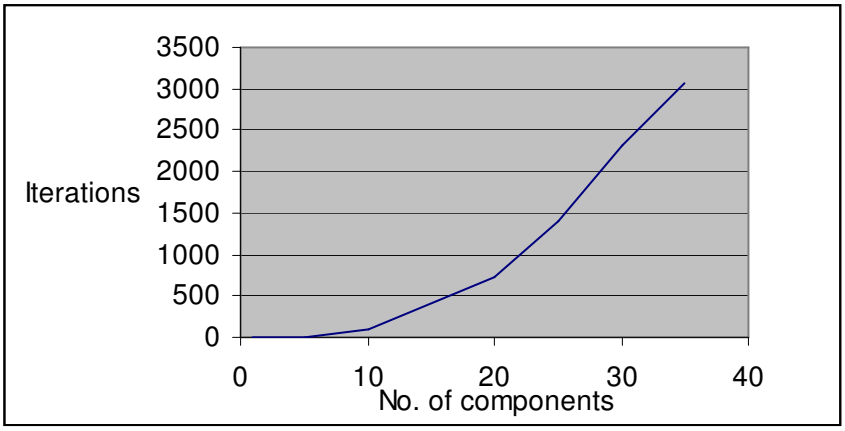


Fig 3.1 A Graph Based Comparisons Between Components and Iterations

The genetic algorithms are used to solve this problem. For effectively reusing the existing components the classification and the retrieval of components plays a major role. The genetic algorithm based software component retrieval is based on the similarity between the components. In the classification method, components are grouped together such that the member of same group has at least one property, which the members of other groups do not have. In retrieval method, more relevant components are found by browsing or linear search. Thus it is very important to classify reusable components elaborately and retrieve them accurately for efficient reuse.

Similar components are clustered on the basis of similarity between them. If similar components are grouped together and classified into each cluster automatically, then

the effort required to retrieve components is reduced. Thus if user locate a relevant cluster in browsing or linear retrieval, they can easily find relevant component in it. However if the number of clusters is fixed, some clusters do not contain only similar component but they may contain other components also. Therefore a multi-way clustering method is needed, in which the number of clusters need not be fixed. In cluster based linear retrieval, information retrieval (IR) techniques are used in order to retrieve cluster containing components close to user requirements. The queries are modified to find out the appropriate clusters, which contains the required components.

3.1. Clustering Using Graph Partitions

The clustering of components is based on the idea of graph partitioning. Let G be the graph G = (V, E) having V vertices and E represents the edges. If two components have at least one of the properties common, then these two components will be connected in graph representation. The partitioning of graph is based on multi way clustering, it divides graph into disjoint sets. The disjoint sets represent the clusters of nodes. The objective function is based on ratio cut the main idea is to minimize the objective function. The ratio cut is the cut that generates the minimum ratio among all cuts in graphs. For example, let c(i,j) be the capacity of an edge connecting node i and node j. (A,B) denotes a cut that separates a set of nodes A from its component B. The capacity of this cut is equal to C(AB) $\sum$ i$\subset$A $\sum$j$\subset$ B c(i,j). The ratio of this cut is defined as R(ab)= (c(ab)/$|$ A*B$|$), where A and B denote the cardinalities of subset A and B, respectively. Since multi-way partitioning divides G using this ratio cut, nodes

are gathered into clusters and number of edges crossing the boundary of cluster is minimized. The objective function is [21]

$$F(P) = 1/n * (k-1)\sum |E_i|/|C_i|$$

Using ratio cut is defined to account for cut nets and size balance among the clusters. In this function, n is the number of nodes, k is the number of clusters, $|E_i|$ is the set of edges crossing the boundary of cluster C_i and $|C_i|$ is the size of cluster. The basic idea is to minimize the edges between different clusters and maximizing with in the cluster. GA is applied to multiway clustering so that reusable components are classified into optimized clusters $P^k = \{C_1, C_2, ..., C_k\}$ whose number is not fixed.

3.2. Proposed Model

The software components are grouped together into clusters. Clustering of components depends upon the similarity between the components. So the clustering of components is exact if the similarity between the components is correct and precise. Similarity measure between the components described using informal specifications is not precise and correct. Formal specifications are used to describe the components because they are unambiguous, complete and consistent. If formal specifications are used to describe the component then the similarity measure between the components is more precise and correct. If the similarity is precise then the clustering is exact.

The proposed model is based on the three similarities between the components the structural similarity, the functional similarity and the behavioral similarity. The structural, functional and behavioral specifications of the components are stored in the components repository and based on these specifications similarity is calculated.

3.3. Component Specifications

The reuse of software components depends upon the specifications of the components described in the repository. Every component is made up of zero or more components in the form of concrete classes. Thus a component is self-contained complex entity consisting of subcomponents, classes, common data and common methods. Each part is structurally related to its subcomponent, which is related to outside world via interface. The components are linked together in the form of interactions thus the behavioral properties are classified in the form of interactions. The component specifications depend on the three views of modeling structural model (static model) behavior (dynamic model) and function (functional model)[27].

Structural specifications the static properties of components are described. The structural properties of components describe the components, subcomponents, classes contained and components relationships.

Behavioral Specifications describe the capabilities of components. These capabilities are combinations of multiple behaviors.

Functional specifications describe how the output values are derived from input values by the operations.

A software component is 5-tuple elements having the form

Software component (SC) = (name, set of sub components, class members, signatures, interactions)

Name = unique name given to component,

Sub components = is the set containing the sub components

Class Members = which is further classified as

Member Class = set of member classes

Common Class = set of common classes

Method Class

Attribute Class

Signature is 5-tuple elements

Signature:

Name of operation:

Input Parameters

Output Parameters

Local Variables

Pre Conditions

Post Conditions

Interaction is 3-tuple elements

Interaction Source

Interaction Destination

Behavior

Name of behavior

State

Action

The components are described as follows.

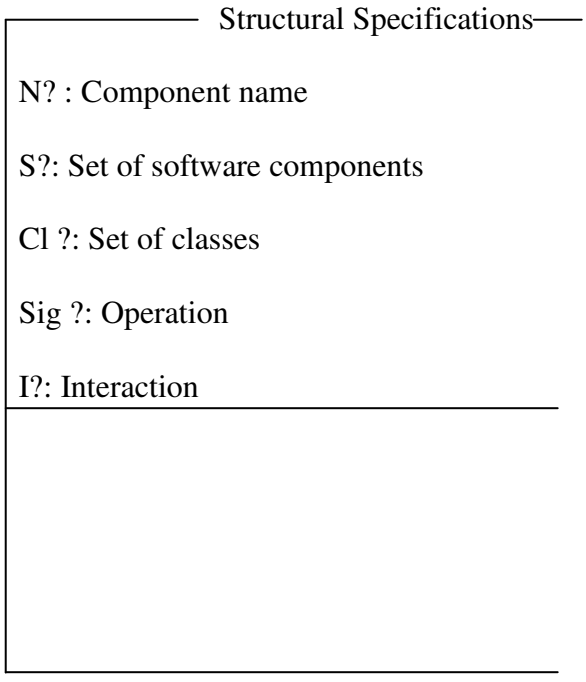
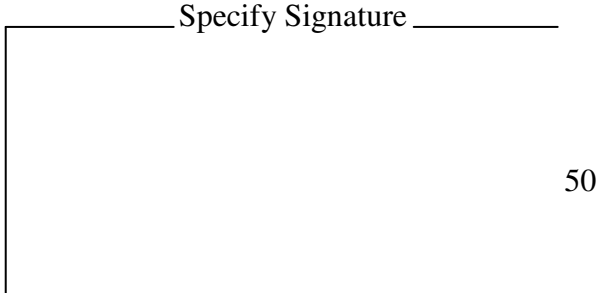


Fig 3.2 Components Structural Specifications



N ?: Operation Name

In ?: Input Parameters

Loc ?: Local Variables

Out ?: Output Parameters

Pre ?: Pre Condition

Post ?: Post Condition

Fig 3.3 Components Signature Specifications

Specify Interaction

Is ?: interaction source

Id ?: interaction destination

B ?: set of behavior

Fig 3.3 Interaction Specifications

Specify Behavior

N ?: behavior name

P?: state

A?: action

Behavioral specifications

Fig 3.4 Behavioral Specifications

3.4. Similarity

The similarity between any two components means that there exists some relation between the components. The similarity between the components lies between 0 and 1 the value 0 means that there is no similarity between the components. The value 1 means there exists a similarity between the components. The similarity between the components is calculated as

Sim (para1, para2) → [0,1]

$$Sim(para1, para2) = \frac{m}{\sqrt{x} \times \sqrt{y}}$$

Where m is the total no of predicates matched

x is the number of predicates in para1

y is the number of predicates in para2

Similarity between the components are given as

Total similarity = (structural similarity + behavioral similarity + functional similarity)/3

The similarity between two components calculated as the average of structural similarity, functional similarity and behavioral similarity.

3.5. Applying Genetic Algorithms in Component Reuse

Genetic Algorithms are applied for clustering of software components based on similarity between the components.

3.5.1. Algorithm for Classification of Components

GA is used to classify the components in the form of clusters and the clustering is multiway clustering i.e. the number of clusters are not fixed. The algorithm for forming the clusters is

a) In order to solve multi-way clustering, a chromosome is formed which consists of genes corresponding to components and the value of gene is the identifier of the cluster to which a component belongs [21]. Thus i^{th} gene indicates the i^{th}

component among N components, a chromosome consists of a N-integer and a solution in which component is assigned to clusters. For example in Fig. 3.6 shows that component 2,3 and 6 are assigned to cluster 3 and component 4,7 and 9 are assigned to cluster 1.

Component	1	2	3	4	5	6	7	8	9	10
Cluster	2	3	3	1	5	3	1	6	1	2

Figure. 3.6 Chromosome Representing Components and Cluster

b) Effectiveness of clustering technique depends on the similarity measure between the components. The similarity between the components is calculated as

Similarity (para1, para2) → returns value between 0 and 1

$$SIM\_BTWN\_COMPS(Comp_i, Comp_j) = TotalSimilarity(comp_i, comp_j)$$

c) Then similarity within the cluster is calculated. The similarity within the cluster is the average of the similarities between the components in that cluster and is defined as follows [21]

$$SIM\_IN\_CLUSTER(Cluster) = \frac{\sum_{Compi,Compj \subset Cluster} SIM\_BTWN\_COMPS(Comp_i, Comp_j)}{\frac{|Cluster| * (|Cluster| - 1)}{2}}, i \neq j$$

Here *Cluster* is the number of components in cluster. In multi-way clustering, this similarity measure for each cluster should be maximized.

d) The similarity measure between two clusters is the average similarity of the corresponding components in the clusters and is defined as follows.

$$SIM\_BTWN\_CLUSTERS(Cluster_i, Cluster_j) = \frac{\sum_{i,j}^{m,n} SIM\_BTWN\_COMPS(Comp_i, Comp_j)}{m \times n}$$

Here m is the number of components in $Cluster_i$ and n is the number of components in $Cluster_j$

e) Total similarity between each formed clusters is defined as follows [21].

$$SIM\_BTWN\_OTHERS(Cluster_j) = \sum_{j-1 j \neq 1}^{k} SIM\_BTWN\_CLUSTERS(Cluster_i, Cluster_j)$$

In a multi-way clustering, this measure should be minimized.

f) Then the objective function is calculated which is based on that the similarity between the clusters should be minimized and similarity within the cluster should be maximized [21].

$$Obj(P^k) = \frac{1}{n \times (k-1)} \frac{\sum^{k}_{i-1} (SIM\_BTWN\_OTHERS(Cluster_j))}{|Cluster_j|} - (SIM\_IN\_CLUSTER(Cluster_j))$$

g) Apply genetic operations on the chromosome.

h) Repeat steps 'a' to 'g' till the chromosome, which minimizes the objective function, is obtained.

3.5.2 Algorithm For Retrieval Of Components

a) The user is given unweighted structural specification, functional specification and behavioral specification of the required component. These specifications are assigned weights randomly. A chromosome is formed of the weight assigned to specifications.

Specifications	Structural	Functional	Behavioral
Weights	0.3…0.4	0.3…0.4	0.3…0.4

Figure 3.7 Chromosome for Retrieval

b) The stored components are assigned weights according to the equivalence classes. A cosine function is used to calculate the similarity of the query component with the cluster. The motive behind the cosine function is that if the component is similar to the cluster that means the angle between them is zero and gives similarity as 1 [21].

$$COSINE(Cluster, Query) = \frac{\sum_{f \subset Facted} \sum^{ti}{}_{k-1} (CTERM_{fk} \times QTERM_{fk})}{\sqrt{\sum_{f \subset Facted} \sum^{ti}{}_{k-1} (CTERM_{fk})^2 \times \sum_{f \subset Facted} \sum^{ti}{}_{k-1} (QTERM_{fk})^2}}$$

c) The objective function of the cluster, which has high value of cosine function, is calculated. The objective function is as follows

$$obj(\ ) = 1 - \frac{2 \times P \times R}{P - R}$$

d) Repeat steps ‘a’ to ‘c’ till the cluster, which minimizes the objective function, is retrieved. The components corresponding to retrieved cluster are shown to user.

Chapter 4

Conclusion and Future Scope

Software Reuse has potential for improving software quality and programmer productivity. Problems, which limit the practice of reuse, are effectively structured software repository and a mechanism for effectively retrieving from this repository. Software repository composed of software components represented by natural language may hinder the retrieval process due to the problem of ambiguity, incompleteness and inconsistency. The components represented with formal specifications do not have these problems. The approach followed in the present work uses Genetic Algorithms with Formal Specifications for clustering the software components, in which the

number of clusters, similarity in cluster and similarity between clusters are taken into consideration with the aim of finding optimized clusters. For genetic algorithms based retrieval the aim is to find the optimal query which retrieve clusters containing components similar to the given query.

4.1. Conclusion

The approach based on genetic algorithm uses structural, functional and the behavioral specification of the component. The component specifications depend on the three views of modeling structural model, behavioral model and functional model. For Similarity calculation the library composed of seven components is considered. In Table 4.1 specifications of only two components are shown.

Table 4.1 Component Specifications

		Component C1	Component C2
Structural Specifications	Component Name	Array-oper	Stack-oper
	Sub Components	Array	Stack , Tree
	Classes	Array, Concrete	Stack, Concrete
	Operation	Add , Delete	Pop
	Transactions	Addition, Deletion	Deletion
Signature Specifications	Name of operation:	Add	Pop
	Input Parameters	A? : Array , I? : Int	S? : Stack , I? : Int
	Output Parameters	A! : Array	S! : Stack
	Local Variables	H : Int	H : Int
	Pre Conditions	I < Max	I >0

	Post Conditions	I = I+1	I = I-1
Functional Specifications	Interaction Source Interaction Dest Behavior	Array Array Adding, Deletion	Stack Stack Deletion
Behavioral Specifications	Name of behavior State Action	Adding Adding, Added Additem	Deleting Deleting, Deleted DeleteItem

The similarity between the components C1 and C2 is calculated as follows

$$StructualSimilarity = 0 + 0 + 1/\sqrt{2 \times 2} + 0 + 1/\sqrt{2} + 0 + 2/\sqrt{4 \times 4} + 0 + 1 + 0 + 0$$

$$= 2.7$$

$$FunctionalSimilarity = 0 + 0 + 1/\sqrt{2 \times 1} = 0.7$$

$$BehavioralSimilarity = 0 + 0 + 0 = 0$$

$$TotalSimilarity = \frac{2.7/11 + 0.7/3 + 0/3}{3} = 0.18$$

Like this way the similarities between other components are calculated.

Table 4.2 Similarities Between Components

Comp 1	Comp 2	Similarity
1	2	0.32
1	3	0.21
1	4	0.64
1	5	0.36
1	6	0.26

1	7	0.32
…….. .	…….	……..
5	6	0.38
5	7	0.26
6	7	0.18

A chromosome is formed representing the components and their corresponding clusters. Genetic operations are performed on the chromosome (CH). After 100 iterations the resulting chromosome is obtained.

The objective function is calculated as below.

N= number of components =7

K= number of clusters for CH1 = 5

$$obj\,(CH\,1) = \frac{1}{7\times 4}(6.39/19 - 0.56) = -0.0079$$

$$obj(CH100) = \frac{1}{7\times 4}\left(5.32/19 - 1.45\right) = -.0.047$$

Thus the chromosome CH100 is selected which minimizes the objective function.

Table 4.3 Chromosome Formed

	C1	C2	C3	C4	C5	C6	C7
CH 1	2	1	0	3	0	3	6
CH2	0	3	6	2	1	0	3
.....	.....	.....	.....	.....	.....	.....	.....

CH 100	2	1	0	2	1	6	3

Thus the C1, C4 and C2, C5 are clustered together based on their similarity value. The formed clusters are assigned weight.

Table 4.4 Weights Assigned

	Structural Weight	Functional Weight	Behavioral Weight
Cl2	0.4	0.4	0.42
Cl1	0.3	0.35	0.35
Cl0	0.35	0.3	0.32
Cl6	0.4	0.44	0.32
Cl3	0.35	0.44	0.45

For retrieving the component a user query is entered for matching. Then specifications in the query are assigned weights randomly. These weights are modified and the resulting chromosome of the modified query is formed.

Table 4.5 Chromosome For Retrieval

	Structural Weight	Functional Weight	Behavioral Weight
CH1	0.35	0.30	0.35
CH2	0.20	0.65	0.35
.....	.....	.....	.....
CH(Final)	0.30	0.30	0.15

The objective function for the chromosome CH1 is calculated as below.

1) After applying Cosine function the cluster 0 gives maximum value.

2) Precision =0/3=0, Recall = 0/0=0

3) Obj(CH1)= 1-0=1

For CH(100) the value of objective function is

$$Obj(CH100) = 1 - \frac{2 \times 0.66 \times 1}{1 + 0.66} = -0.204$$

The resulting cluster number 2 is retrieved means component C1 and C4 are retrieved. The similarity between the query component and the resulting cluster is

Table 4.6 Similarity of Query Component to the Stored Components

	C1	C2	C3	C4	C5	C6	C7
Query Comp	0.84	0.21	0.12	0.62	0.06	0.42	0.17

The effectiveness of the retrieval process is measured in the terms of precision and recall and is

Relevant Component = If(Sim(Cx,Cy)>0.4)= 3

Precision = 2/3 = 0.6

Recall = 2/2 = 1

The benefits of this approach over the approach using natural language specification are

- The formal specifications remove the ambiguity, incompleteness and inconsistency.
- The formal specifications focus on the semantic behavior rather than syntactic behavior i.e. it emphasis on what and not how.
- The similarity between the components specified in formal specification is more precise than the similarity between the components specified using natural language.
- The clustering is exactly based on similarity value.
- Effectiveness of retrieval process depends on clustering

4.2 Future Scope

Future scope includes extending this work.

- This approach is combined with the type checker Z/Eves to verify the correctness of the specifications.
- This approach can be combined with other soft computing technique to improve the retrieval effectiveness.
- It could be enhanced to search more than one behavioral schema and more than one functional schema.

References

[1] T. Kohonen. “Self-organized formation of topologically correct feature maps”. Biological Cybernetics 43. (1982).

[2] Salton G. And Buckley,” Term-weighting approaches in automatic text retrieval”. Inf. Process. Manage. 24, 5, 513–523 (1988).

[3] T. Kohonen. “Self-Organization and Associative Memory (3rd edition)”. Berlin: pringer.(1989).

[4] T. Kohonen.” The Self-Organizing Map”. Proceedings of the IEEE 78(9).(1990)

[5] Basili, V. R. And Rombach, H. D. “Support for comprehensive reuse”. Software . Eng. J. 6, 5, 303–316 (Sept. 1991).

[6] Ruben Prieto-Diaz “Implementing Faceted Classification for Software Reuse” Software Production Consortium, Herndon, VA. ACM Press New York, NY, USA; Pages: 88 - 97: (1991)

[7] Kevin Cox “Information Retrieval By Browsing “Proceedings of The 5th International Conference on New Information Technology HongKong University of Science & Technology Kowloon, HongKong (Dec 1992).

[8] PRIETO-DÍAZ, R.. “Status report: Software reusability’. IEEE Softw. 10, 3 , 61–66 (May 1993).

[9] Jun-Jang Jeng and Betty H.C. Cheng ``Using Formal Methods to Construct a Software Component Library,” Lecture Notes in Computer Science, Vol. 717, Springer-Verlag, pp. 397--417, %in Proc. of Fourth European Software Engineering Conference, (September 1993).

[10] Zaremski, M and Wing, J. M “Signature matching: A key to reuse”. Technical Report CMUCS-93-151, Carnegie Mellon University, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, (May 1993).

[11] Khuzaima S. Daudjee Thesis report on “Organizing Reusable Software Repositories Through Heuristic Clustering’. Department of Computer Science York University Canada (October 1994).

[12] ``Specification Matching for Software Reuse: A Foundation" (, Proc. of ACM Symposium on Software Reuse , Seattle, Washington,(April 1995).

[13] M. R. Girardi, "Classification and Retrieval of Software through their Descriptions in Natural Language", Ph.D. dissertation, No. 2782, University of Geneva, (December 1995)

[14] Steven Atkinson, Roger Duke “Behavioral Retrieval from Class Libraries “ (1995) .

[15] Damiani, E. And Fugini “ Fuzzy identification of distributed components”. In Computational Intelligence—Theory and Applications, B. Reusch, Ed. Springer Lecture Notes in Computer Science, vol. 1226 M. G. (1997).

[16] Rogger S. Pressman. ‘Software Engneering A Practitioner’s Approach” Fourth Edition (1997).

[17] Betty H.C. Cheng and Jun-Jang Jeng ‘‘Reusing Analogous Components” , in IEEE Trans. on Knowledge and Data Engineering, Vol 9., No. 2, , pp. 341--349. (March/April 1997).

[18] Chao-Tsun Chang, William C. Chu, Chung-Shyan Liu, Hongji Yang: “ A formal approach to software components classification and retrieval.” 264-269 Electronic Edition (IEEE Computer Society DL) Colin J. Hardy, Hlen M. Edwards, J. Barrie Thompson (1997).

[19] P. Chen, R. Hennicker, and M. Jarke." On the retrieval of reusable software components" (August 11-15, 1997).

[20] Sommerville, Ian. “Software Engineering. ‘Addison Wesley.157-165 (1998).

[21] Byung-Jeong Lee, Byung-Ro Moon .” Optimization of Multi-Way clustering and Retrieval Using genetic Algorithms in Reusable Class Library “ (1998).

[22] M.~Sitaraman and G.~Leavens (with Y. Chen) ``A Semantic Foundation for Specification Matching" in Foundations of Component-Based Systems Eds., Cambridge University Press, (2000).

[23] Spivey, J.M. “An introduction to Z and Formal Secifications” Published in Software Engineering Journal Volume: 4, Issue: 1 On page(s): 40-50 (Jan 1989).

[24] Cybulski, J. “Application of software reuse methods to requirement elicitation from informal requirements texts”. Ph.D. thesis2001.

[25] Khayati, Oualid and Giraudin Jean Pierre “Component Retrieval System” (2001).

[26] Jingbo Zhu, Tianshun Yao: “A Knowledge-based Approach to Text Classification”.(2001).

[27] Sathit Nakkrasae and Peraphon Sophatsathit, “A Formal Approach for Specification and Classification of Software Components” *Proceedings of the 14th International Conference on Software Engineering and Knowledge*

Engineering, in cooperation with ACM-SIGSOFT. Ischia, Italy, (July 15-19, 2002), pp. 773-780.

[28] Jeff Plummer "Introduction to Genetic Algorithms ". Nov. 2003

[29] Sathit Nakkrasae, Peraphon Sophatsathit " Fuzzy Subtractive Clustering Based Indexing Approach for Software Components Classification" International Journal of Computer & Information Science, Vol. 5, No. 1, (March 2004) .

Sites

[1] www.dacs.dtic.mil/techs/fmreview/intro.html - 8k

[2] hissa.nist.gov/~black/Papers/formMeth.html

[3] http://www.comp.lancs.ac.uk/computing/research/cseg/projects/APPRAISAL/paper/paper.html

[4] http://www.e-papyrus.com/hypertext_review/chapter5.html

[5] http://www.umcs.maine.edu/~ftp/wisr/wisr8/papers/atkinson/atkinson.html

Paper Accepted/Communicated

1. Navjoti Pandhi, "Soft Computing Techniques in Software Reuse" published in Proceedings of National Conference on Software Engineering Process and Practices (SEPP' 04) CSED, T.I.E.T., Patiala (5-6 March), 139-145.
2. Navjoti Pandhi, Rajesh K. Bhatia, Dr. Mayank Dave and Dr. R C Joshi, " A Review of Reusable Component Retrieval Techniques" communicated to CSI Journal (2004).
3. Navjoti Pandhi, Rajesh K. Bhatia, Dr. Mayank Dave and Dr. R C Joshi, " A Review and Comparative Analysis of Reusable Component " communicated to International Journal of Computational Intelligence (IJCI) Journal (April 2004